**UNIVERSIDADE FEDERAL DE SANTA CATARINA**

**PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**

**Neiva Scheidt**

# INTRODUÇÃO DE SUPORTE GERENCIAL BASEADO EM *WORKFLOW* E CMM A UM AMBIENTE DE DESENVOLVIMENTO DE *SOFTWARE*

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Ricardo Pereira e Silva

Orientador

Florianópolis, Março de 2003.

# INTRODUÇÃO DE SUPORTE GERENCIAL BASEADO EM *WORKFLOW* E CMM A UM AMBIENTE DE DESENVOLVIMENTO DE *SOFTWARE*

Neiva Scheidt

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Fernando A. O. Gauthier
Coordenador

_____

Prof. Ricardo Pereira e Silva
Orientador

Banca Examinadora

_____

Profa. Nina Edelweiss

_____

Prof. Leandro José Komosinski

_____

Prof. Rosvelter Coelho da Costa

# SUMÁRIO

# LISTA DE ACRÔNIMOS

| | |
|---|---|
| CMM | *Capability Maturity Model* |
| KPA | *Process Area* |
| SPICE | *Software Process Improvement and Capability Determination* |
| SQA | Garantia de Qualidade de *Software* |
| UML | *Unified Modelling Language* |
| WFMS | *Workflow Management Systems* |
| WfMC | *Worfkflow Managment Coalition* |

# LISTA DE FIGURAS

# LISTA DE TABELAS

# RESUMO

*É comum entre os ambientes de desenvolvimento de software o apoio ao aspecto tecnológico do desenvolvimento, sem proporcionar suporte ao gerenciamento dos processos de produção. A tecnologia de Workflow mostra-se adequada à modelagem dos processos envolvidos em um desenvolvimento de software. CMM (Capability Maturity Model) estabelece requisitos a serem contemplados nos processos no desenvolvimento de software.*

*O presente trabalho explora conjuntamente Worflow e CMM para suportar o gerenciamento em um ambiente de desenvolvimento de software. O ambiente SEA serve como objeto de experiência para a inclusão de modelagem e gerenciamento de Workflow e inclusão de restrições de CMM para auxílio ao gerenciamento dos processos. A idéia fundamental é modelar os processos do desenvolvimento de software usando uma ferramenta de modelagem de Workflow e usar essa modelagem ao longo do desenvolvimento, em tempo real, para suportar o gerenciamento. Esta ferramenta de Workflow conta com requisitos de CMM em sua estrutura. Os resultados obtidos com um protótipo desenvolvido são apresentados na conclusão desse estudo.*

***Palavras-chave**: Workflow, CMM, framework, ambientes de desenvolvimento de software, modelagem de processos.*

# ABSTRACT

*It is common for software development environment to help the technological aspect of development, without propose management process sustain. Workflow technology is proper to be used for modeling the software development. CMM (Capability Maturity Model) establishes some rules that can be submitted during the software development process.*

*This research explores Workflow and CMM as a management support in a software development environment. SEA is used as an experiment of the Workflow modeling inclusion, Workflow management and CMM rules to assist the process management. The main idea is to model the processes of software development using a Workflow tool modeling and use this modeling during the software development. The development uses Workflow Management in runtime, to support management. This Workflow tool has CMM on its structure. The prototype results are shown in the conclusion of this work.*

***Key-words**: Workflow, CMM, framework, software development environment, process modeling.*

# 1 INTRODUÇÃO

## 1.1 Considerações Gerais

A automatização de processos fornece meios para integrar pessoas em uma organização de desenvolvimento de *software* com o processo de desenvolvimento e as ferramentas que suportam esse desenvolvimento. Essa tecnologia tem o potencial de melhorar significativamente a qualidade e a produtividade do *software*.

Sistemas de *software* têm crescido em termos de tamanho e complexidade e com a modelagem do processo de desenvolvimento existe um melhor entendimento sobre os passos a serem seguidos durante um projeto. Abordada da maneira correta, a automatização do processo tem a habilidade de melhorar a qualidade e produtividade do desenvolvimento de *software* (ARAUJO, 1998).

A preocupação com os ambientes de desenvolvimento de *software* também está associada à forma com que as atividades são gerenciadas e controladas dentro de uma organização.

O foco deste trabalho é o problema do planejamento, acompanhamento e supervisão durante o processo de desenvolvimento de um *software*. Tipicamente os ambientes de desenvolvimento de *software* suportam a produção do *software*, mas não o gerenciamento desta produção. Este trabalho apresenta uma proposta de modelagem e gerenciamento de processos em um ambiente de desenvolvimento de *software*.

## 1.2 Ambiente de Desenvolvimento de *Software*

Um ambiente de desenvolvimento de *software* é um sistema que apóia o processo de desenvolvimento de *software* de forma geral, seguindo um paradigma de desenvolvimento de *software*.

SEA é um ambiente de desenvolvimento de *software,* produto da tese de doutorado de (SILVA, 2000), que permite o desenvolvimento de aplicações usando as

técnicas de modelagem UML, possibilitando a utilização integrada das abordagens de desenvolvimento orientado a componentes e desenvolvimento baseado em *frameworks*. Estas abordagens têm em comum a ênfase em promover reuso, tanto de implementação, como de projeto. SEA suporta o desenvolvimento de *frameworks*, componentes e aplicações como estruturas baseadas no paradigma de orientação a objetos (SILVA, 2000a).

SEA foi desenvolvido sob o *framework* OCEAN (SILVA, 2000a). O *framework* OCEAN constitui um suporte flexível e extensível para a construção de ambientes de desenvolvimento de *software*. É caracterizado pela capacidade de suportar a construção de ambientes que manipulem diferentes tipos de especificação e que disponham de diferentes conjuntos de funcionalidades.

## 1.3 *Workflow*

O estudo de *Workflow* é apresentado com maior profundidade no capítulo 5, mas para um melhor entendimento dos objetivos deste trabalho, é feita uma breve introdução a este assunto.

*Workflow* pode ser definido como uma coleção de tarefas organizadas para realizar um processo de negócio. Uma tarefa pode ser executada por um sistema de computador, por um agente humano ou pela combinação desses dois. (AMARAL, 1997)

*Workflow* pode modelar processos[1] que definem a ordem de execução e as condições pelas quais as tarefas[2] são iniciadas, representando a sincronização das tarefas e o fluxo de informações. Com o uso de *Workflow*, documentos, informações e tarefas são passadas de um participante para outro, a fim de que sejam tomadas ações de acordo

---

[1] Um processo compreende um conjunto de uma ou mais atividades estruturadas que, coletivamente, realizam um objetivo no contexto de uma estrutura organizacional.

[2] Uma tarefa é uma fração do processo.

com um conjunto de regras e procedimentos. *Workflow* significa fluxo de trabalho, algo distinto de fluxo de dados (THOM & SCHEIDT, 1999).

## 1.4 Qualidade do Processo de Desenvolvimento de *Software*

O controle dos processos no desenvolvimento de um *software* é um pré-requisito para o estabelecimento de um programa de garantia de qualidade.

O processo de desenvolvimento controlado inclui a definição de um ciclo de desenvolvimento, as atividades envolvidas no mesmo, os métodos e ferramentas a serem utilizados para realizar cada uma destas atividades e os procedimentos de gerência do processo e do controle da qualidade (ARAUJO, 1998).

Os métodos de avaliação do processo de *software* caminham geralmente para o gerenciamento do desenvolvimento de *software.* A modelagem, evolução e monitoramento do processo são identificados como as funções mais importantes para a melhoria desse gerenciamento.

Dentre os modelos e normas de qualidade de processos de *software* existentes, alguns, dos mais conhecidos, foram estudados para a escolha do modelo a ser seguido: Norma ISO/IEC 12207, SPICE e CMM.

A Norma ISO/IEC 12207 objetiva estabelecer uma estrutura comum para os processos de ciclo de vida de *software*, com terminologia bem definida, para ser utilizada por desenvolvedores para construir, gerenciar e melhorar *software*. A estrutura contém processos, atividades e tarefas a serem aplicados durante a aquisição, fornecimento, desenvolvimento, manutenção e operação de produtos de *software*, e que devem ser adaptados ao contexto e características de cada projeto de *software*. Tal adaptação consiste na exclusão de processos, atividades e tarefas não aplicáveis. Com o objetivo de evitar conflitos com procedimentos já adotados pela organização, o processo de adaptação deve estabelecer uma compatibilidade com políticas e normas já existentes na organização. (ZAHRAN, 1998; ROCHA *et al*., 1999)

A Norma ISO/IEC 12207 agrupa os processos de ciclo de vida em três grandes classes: processos fundamentais, processos de apoio e processos organizacionais. Cada processo de ciclo de vida está dividido em um conjunto de atividades e cada atividade em um conjunto de tarefas.

O Modelo SPICE (BARRETO, 1997) constitui-se de um padrão para a avaliação do processo de *software*, visando determinar a capacitação de uma organização. A norma visa ainda orientar a organização para uma melhoria contínua do processo, atendendo alguns aspectos da Qualidade do Processo de *Software*, como planejamento do projeto de *software*, gerenciamento dos processos e monitoração do andamento dos processos.

O SPICE inclui um modelo de referência, que serve de base para o processo de avaliação. Este modelo é um conjunto padronizado de processos fundamentais, dividido em cinco grandes categorias de processo: Cliente-Fornecedor, Engenharia, Suporte, Gerência e Organização. Cada uma destas categorias é detalhada em processos mais específicos. Além dos processos, o SPICE define também os seis níveis de capacitação de cada processo: incompleto, executado, gerenciado, estabelecido, previsível e otimizado. Cada um dos processos deve ser classificado nestes níveis. (BARRETO, 1997)

*Capability Maturity Model* (PAULK et.al., 1994) descreve princípios e práticas dos quais depende a maturidade do processo de *software*. CMM constitui-se de um conjunto de metas a serem alcançadas durante o desenvolvimento de *software*, contando com práticas que devem ser cumpridas para a obtenção de qualidade de processo. Seu objetivo é auxiliar as organizações a aumentarem a maturidade de seu processo por um caminho evolutivo. CMM pode ser utilizado como um modelo de referência para a melhoria do processo de *software*.

## 1.5 Objetivo do Trabalho

O objetivo deste trabalho é inserir suporte de gerenciamento no ambiente SEA. Visando alcançar esse objetivo, é proposta a união de *Workflow* e CMM às

funcionalidades do ambiente SEA. Assim, o novo ambiente de desenvolvimento de *software*, além das funcionalidades já disponíveis, passa a ter a possibilidade de modelar os processos e gerenciar os processos de acordo com os requisitos de gerenciamento propostos.

*Workflow* é um mecanismo que tem se mostrado adequado para modelar o aspecto gerencial de processos. A idéia é automatizar os processos através de uma ferramenta que modele *Workflow* e o use ao longo do desenvolvimento como suporte ao gerenciamento.

CMM foi o modelo escolhido para o ambiente SEA porque é adequado ao uso de ambientes de desenvolvimento de *software* e as KPAs (atividades que auxiliam os processos em um desenvolvimento) do nível 2 de CMM perfazem as etapas de gerenciamento de um projeto, proporcionando documentação de estimativas, planejamento e compromissos estabelecidos. Essas características o ambiente SEA não possui. Agregadas a *Workflow,* as restrições de CMM podem prestar qualidade ao desenvolvimento de *software* no ambiente SEA.

O ambiente SEA foi o escolhido como objeto de experiência por não apresentar gerenciamento, não apresentar modelagem de seus processos e por estar disponível a novas inclusões e alterações na sua estrutura.

## 1.6 Estrutura da Dissertação

Essa dissertação está organizada em oito capítulos, descritos a seguir:

O Capítulo 2 revisa conceitos e características do desenvolvimento de *software* baseado em *frameworks*. Este estudo faz-se necessário para a compreensão do uso de *framework*s, visto que o *framework* OCEAN foi a base para a criação do ambiente SEA e para o suporte de *Workflow*. Usando os recursos de extensão do *framework* OCEAN, novas funcionalidades podem ser introduzidas ao ambiente de desenvolvimento de *software*.

O Capítulo 3 apresenta o ambiente de desenvolvimento de *software* SEA e o *framework* OCEAN. A parte referente ao *framework* OCEAN apresenta o suporte a criação de estruturas de documentos, o suporte à edição semântica de especificações e a estrutura de suporte à composição de ações de edição. A parte referente ao ambiente SEA apresenta sua estrutura de especificação: especificação OO (especificação de artefato de *software* baseada no paradigma de orientação a objetos e produzida a partir do uso de notação de alto nível – UML), especificação de interface de componente e *cookbook* ativo (para orientar o uso de um *framework* previamente desenvolvido).

O Capítulo 4 descreve CMM – *Capability Maturity Model* – dando ênfase ao nível 2. CMM propõe um conjunto de requisitos a serem seguidos, objetivando qualidade em processos. A ênfase desse capítulo é apresentar as características de CMM, os níveis de maturidade de CMM, suas KPAs e, detalhar as KPAs do nível 2, que são compostas de atividades a serem cumpridas durante o desenvolvimento de um processo.

O Capítulo 5 introduz *Workflow*, apresenta suas características, vantagens e o seu meta-modelo. É descrito um exemplo de modelagem de *Workflow* que é indicada como modelo para os processos do desenvolvimento de *software*.

O Capítulo 6 apresenta a inclusão de *Workflow* ao ambiente SEA. São discutidas as etapas de adesão do Editor de *Workflow* ao ambiente, onde e como o gerenciamento atua no modelo de processos, e como as regras de CMM podem auxiliar o andamento das atividades.

O Capítulo 7 apresenta um exemplo de desenvolvimento de *software* usando o ambiente SEA2, o novo ambiente de desenvolvimento de *software* que possui todas as características do ambiente SEA mais gerenciamento de processos. Com este exemplo é possível criar um projeto e acompanhar seus processos (através do *Workflow*) até a sua conclusão, verificando assim, as novas ferramentas do ambiente SEA2.

As conclusões da dissertação e as perspectivas para trabalhos futuros encontram-se descritas no Capítulo 8.

# 2 *FRAMEWORKS* ORIENTADOS A OBJETOS

O presente capítulo caracteriza os conceitos de *frameworks*, visto que o ambiente de desenvolvimento de *software* SEA foi desenvolvido sob o *framework* OCEAN.

## 2.1 Definição de *Framework*

A realidade da indústria de *software* é que os programas estão cada vez mais complexos e está sendo exigido cada vez menos tempo para seu desenvolvimento. Quanto mais rápida a sua conclusão, menor o seu custo. Uma solução para isso pode ser o reuso de *software*. O reuso de artefatos de *software* pode ser obtido através de rotinas, módulos, componentes, padrões ou *framework*s.

Um *framework* é um esqueleto de implementação de uma aplicação ou de um subsistema de aplicação, em um domínio de problema particular. É composto de classes abstratas e concretas e provê um modelo de interação ou colaboração entre as instâncias de classes definidas pelo *framework*. Um *framework* é utilizado através de configuração ou conexão de classes concretas e derivação de novas classes concretas a partir de classes abstratas do *framework.* (WIRFS-BROCK, 1990)

A diferença fundamental entre o reuso de um *framework* e a reutilização de classes de uma biblioteca, é que neste caso são usados artefatos de *software* isolados, cabendo ao desenvolvedor estabelecer sua interligação, e no caso do *framework*, é procedida a reutilização de um conjunto de classes inter-relacionadas, cujo inter-relacionamento é estabelecido no projeto do *framework*. (SILVA, 2000a)

O *framework* constitui-se de uma estrutura de classes ou uma implementação propositalmente inacabada. Ele surge da idéia de generalizar um domínio onde, a partir de várias aplicações com características em comum, cria-se uma especificação geral para ser usada por todas elas, ou seja, cria-se um *framework* para este domínio.

Ao usar um *framework* de um domínio existente, pode-se alterar suas funcionalidades via sobreposição (por exemplo, herdar um método do *framework* e

modificá-lo na aplicação) e, assim, produzir uma nova aplicação a partir de um *framework* já pronto. A estrutura de classes é adaptada para a geração de uma aplicação específica. Um *framework* pode originar não apenas uma outra aplicação, mas também outros tipos de artefatos de *software*, como por exemplo, outro *framework*.

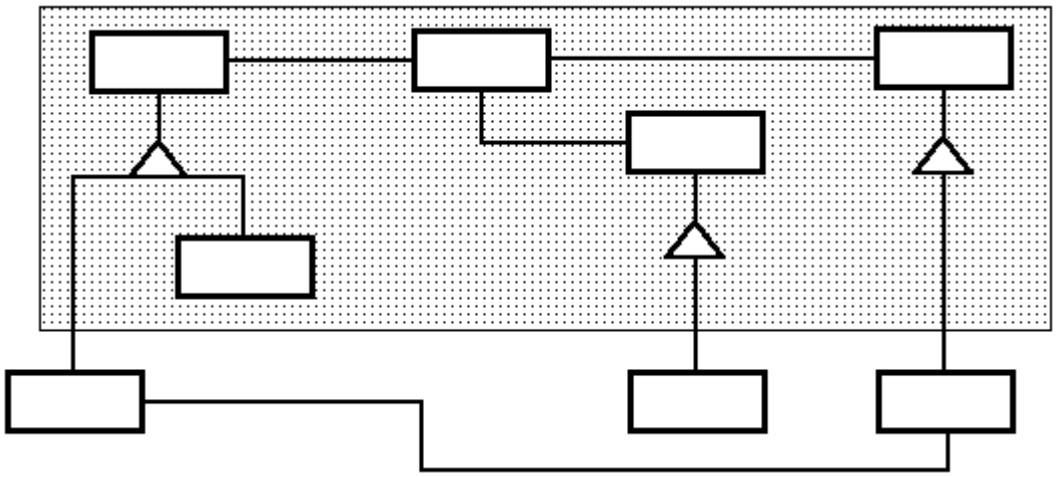


Figura 2.1 – Estrutura de um *framework* (SILVA, 2000a)

A Figura 2.1 mostra como pode ser produzida uma aplicação baseada em um *framework*. São utilizadas as classes do *framework* (que estão dentro do retângulo) e incluídas as classes do projeto a ser realizado. É definida apenas parte dos relacionamentos entre as classes, pois uma parte das interligações já vem pronta.

(SILVA, 2000a apud TALIGENT, 1995) descreve dois aspectos que caracterizam um *framework*: *frameworks* fornecem infraestrutura de projeto (interconcexões pré-estabelecidas definem a arquitetura da aplicação, liberando o desenvolvedor desta responsabilidade, diminuindo a quantidade de código a ser desenvolvida) e *frameworks* "chamam", não são "chamados" (em tempo de execução, as instâncias das classes desenvolvidas esperam ser chamadas pelas instâncias das classes do *framework*). Esta segunda característica é conhecida como "*Princípio de Hollywood*".

*Frameworks* estão ligados diretamente a alguns conceitos:

- Domínio de Aplicações: consiste de uma família de aplicações com características comuns. Está ligado diretamente à *framework*s, pois os *framework*s são desenvolvidos para domínios específicos, agrupando as características comuns.

- Análise de Domínio: processo de identificação e organização de conhecimento a respeito de uma classe de problemas para suportar a descrição e solução destes problemas. O desenvolvimento de um *framework* é um processo específico de análise de domínio.

- Modelo de Domínio: descrição de um domínio através de alguma linguagem. Um *framework* é um modelo de domínio que usa uma linguagem de programação orientada a objetos e, eventualmente, uma linguagem de especificação orientada a objetos.

## 2.2 Características

Um *framework* se destina a gerar diferentes aplicações para um domínio e, portanto, precisa conter uma descrição dos conceitos deste domínio. As classes abstratas de um *framework* são os repositórios dos conceitos gerais do domínio de aplicação. Neste contexto, o método de uma classe abstrata pode ser deixado propositalmente incompleto para que sua definição seja acabada na geração de uma aplicação. Apenas atributos a serem utilizados por um domínio são incluídos em classes abstratas.

*Framework*s são considerados flexíveis, pois mantêm flexível parte de sua estrutura de classes. A flexibilidade permite que a estrutura do *framework* que está sendo usada seja especializada, produzindo diferentes aplicações.

A flexibilidade é a possibilidade de alteração de parte da estrutura de classes do *framework*. Esta alteração é feita pela modificação da estrutura existente ou pela inserção de novos elementos à estrutura. É possível flexibilizar alterando o *framework* ou estendendo o *framework*. A possibilidade de modificar a estrutura é chamada de alterabilidade e a possibilidade de inserir novos elementos que não existem é chamada de estendibilidade. Este conjunto é a idéia de flexibilidade.

A característica de generalização leva muitas vezes os *frameworks* a se tornarem difíceis de compreender, bem como de utilizar. Isto porque, para atender a um determinado domínio de problema ou permitir o desenvolvimento de um conjunto de aplicações diferentes, os *frameworks* necessitam abranger um grande conteúdo dos mesmos. (KLABUNDE, 2000)

Um exemplo de *framework* é o *framework* OCEAN que serve para desenvolver ambientes de desenvolvimento de *software*. No capítulo 3 este *framework* é detalhado.

## 2.3 Classificação de *Frameworks*

Os *framework*s podem ser classificados de acordo com o seu uso, dimensão e finalidade. A seguir estas classificações são descritas.

*Frameworks* de acordo com o seu uso:

- *Framework* tipo caixa-branca: permite ao usuário visualizar como o *framework* foi implementado e, a partir de classes já existentes (através de herança), criar no *framework* novas subclasses das classes abstratas adicionando funcionalidades particulares de sua aplicação.

- *Framework* do tipo caixa-preta: apenas especificam, através de interfaces, os serviços que são por eles disponibilizados e requisitados. Facilitam ao usuário o seu uso, mas reduzem a flexibilidade. O usuário não é totalmente livre para criar ou alterar funcionalidades a partir da criação de classes, uma vez que o controle da aplicação é definido nas classes do *framework* e não nas classes do usuário (SILVA, 2000a). Não permite a criação de subclasses e as aplicações são construídas a partir das classes do *framework*.

- *Frameworks* do tipo caixa-cinza: já tem partes prontas e permite que se insiram novas funcionalidades a partir da criação de classes. Este tipo de *framework* fornece subclasses concretas e permite a criação de novas subclasses concretas. O *framework* OCEAN possui estas características e, portanto, é classificado como caixa-cinza.

*Frameworks* de acordo com sua dimensão (SILVA, 2000a apud DEUTSCH, 1989):

- *Framework* de uma classe: o *framework* contém uma classe parametrizável ou abstrata ou alterável.

- *Framework* de múltiplas classes: o *framework* contém um conjunto de classes interrelacionadas.

*Frameworks* de acordo com sua finalidade:

- *Framework* para geração de aplicações completas: toda a estrutura de uma aplicação é prevista no *framework*.

- *Framework* para suportar a produção de unidades funcionais: *framework* criado para resolver funcionalidades específicas. O *framework* pode prover apenas parte de uma aplicação (interface, armazenamento de dados, etc.).

## 2.4 Vantagens do Uso de *Frameworks*

O uso de *framework*s em desenvolvimento de aplicações apresenta algumas vantagens (SILVA, 2000b). Dentre elas, destacam-se:

- Reuso de trechos de código, de classes, métodos e estruturas já desenvolvidos;
- Reuso de artefatos de *software* já desenvolvidos e depurados;
- Possibilidade de manipulação de classes prontas;
- Possibilidade de usar e alterar uma estrutura já definida;
- Reaproveitamento de concepções arquitetônicas de um artefato de *software* em outro, não necessariamente com a utilização da mesma implementação.

Como conseqüência das características anteriores, há uma tendência de redução do tempo de desenvolvimento e testes, bem como uma tendência de redução de introdução de erros na produção de novos artefatos. Assim, há uma maior qualidade e produtividade na produção de aplicações.

A principal desvantagem do uso de um *framework* é a sua dificuldade de compreensão. É preciso identificar as suas principais classes, já que essas classes fazem, quais métodos são importantes e o que os métodos fazem. As fontes de informação para estes aspectos são conseguidas no código fonte e na documentação do projeto, que nem sempre está disponível, ocupando tempo e esforço do desenvolvedor. (SILVA, 2000b)

## 2.5 Desenvolvimento de Aplicações a partir de *Frameworks*

A finalidade básica de um *framework* é sua reutilização na produção de diferentes aplicações, minimizando tempo e esforço requeridos para isto. Desta forma, procura ser uma descrição aproximada do domínio, construída a partir das informações até então disponíveis (SILVA, 2000a).

O primeiro passo no uso de um *framework* no desenvolvimento de uma aplicação é o aprendizado, por parte do desenvolvedor, de como usar o *framework*. O desenvolvedor precisa conhecer os recursos que o *framework* disponibiliza para evitar esforços desnecessários durante o desenvolvimento da aplicação. (SILVA, 2000b)

Os recursos do *framework* dizem respeito às suas classes - como as classes estão interconectadas - aos seus métodos e ao que cada método faz, para o desenvolvedor saber exatamente onde vai mudar e o quê vai mudar da estrutura existente.

Depois do entendimento do sistema contido no projeto do *framework,* o desenvolvedor da aplicação detalha as particularidades da aplicação específica. A implementação de uma aplicação a partir do *framework* pode ser feita a partir de combinações de instâncias das classes concretas do *framework* ou a partir da definição de novas classes – ou ainda a partir de combinações das duas possibilidades. A possibilidade de usar uma ou outra forma é uma característica particular de cada *framework*, definida em seu projeto. O projeto de um *framework* estabelece a flexibilidade provida, impondo restrições a serem consideradas pelos usuários. Um *framework* pode dispor de documentação específica para ensinar usuários a gerarem aplicações, estabelecendo estas restrições. (SILVA, 2000b)

A utilização ideal de *frameworks* consiste em completar ou alterar procedimentos e estruturas de dados neles presentes. Sob esta ótica, uma aplicação gerada sob um *framework* não deveria incluir classes que não fossem subclasses de classes do *framework*. Todavia, como um *framework* nunca é uma descrição completa de um domínio, é possível que a construção de aplicações por um *framework* leve à obtenção de novos conhecimentos do domínio tratado, indisponíveis durante a sua construção (SILVA, 2000a).

Cabe observar que, além do desenvolvimento de aplicações a partir de *frameworks*, estes também podem servir de exemplo para o desenvolvimento de outros *frameworks* ou fazer parte do desenvolvimento de novos *framework*s (KLABUNDE, 2000).

A compreensão e o uso de *framework*s é de muita importância para este trabalho, pois o ambiente SEA foi construído sob um *framework*, o *framework* OCEAN. O próximo capítulo descreve este *framework* e o ambiente de desenvolvimento de *software* SEA.

# 3 O *FRAMEWORK* OCEAN E O AMBIENTE DE DESENVOLVIMENTO DE *SOFTWARE* SEA

O *framework* OCEAN e o ambiente de desenvolvimento de *software* SEA foram desenvolvidos durante o doutorado de (SILVA, 2000a). Assim, a tese "*Suporte ao desenvolvimento e uso de frameworks e componentes*" é a referência bibliográfica deste capítulo (SILVA, 2000a).

SEA é voltado ao desenvolvimento e uso de *frameworks* e componentes e foi construído sob o *framework* OCEAN. Ambos SEA e OCEAN foram implementados em *Smalltalk*, sob o ambiente *VisualWorks*, reutilizando classes da biblioteca de classes deste ambiente e usando o *framework HotDraw* (SILVA, 2000a apud BRANT, 1995) para suporte ao desenvolvimento de editores gráficos.

No decorrer deste capítulo são apresentados o *framework* OCEAN e o ambiente SEA. Em relação ao *framework* OCEAN são mostrados suas características, sua estrutura e seu suporte quanto à criação de estruturas de documentos. São apresentadas também as estruturas de edição semântica de especificações e composição de ações de edição complexas através de ferramentas.

Em relação ao ambiente SEA, são mostradas suas características, funcionalidades, requisitos disponibilizados, sua estrutura de especificação, os mecanismos de interligação dos seus elementos de especificação e sua estrutura semântica.

## 3.1 *Framework* OCEAN

O *framework* OCEAN é do tipo "caixa cinza", voltado à produção de diferentes ambientes de desenvolvimento de *software*. Ele possibilita que tais ambientes manipulem diferentes estruturas de especificação e apresentem diferentes funcionalidades.

O *framework* OCEAN contém superclasses abstratas (que tratam da definição genérica dos elementos) e classes concretas (que implementam elementos específicos).

Um ambiente específico é produzido a partir de uma subclasse concreta de *EnvironmentManager*. Nesta subclasse estarão definidas as seguintes características:

- O tipo de especificação a ser tratada;
- O mecanismo de visualização e edição associado a cada modelo[3] e conceito[4] tratado pelo ambiente;
- O mecanismo de armazenamento de especificações utilizado pelo ambiente;
- As ferramentas utilizáveis para manipular especificações.

Para suprir a flexibilidade necessária para suportar diferentes conjuntos de técnicas de modelagem, parte da estrutura do *framework* OCEAN corresponde à definição genérica de uma estrutura de documento, a ser estendida para a produção de estruturas específicas. Estas estruturas terão características próprias e flexibilidade para serem alteradas a qualquer momento, pois o *framework* OCEAN foi construído para ter esta capacidade.

A seguir são mostradas as estruturas dos documentos, os mecanismos de visualização dos elementos, o suporte à navegação e os mecanismos de armazenamento de especificações.

---

[3] Um modelo de uma especificação, como um diagrama de classes por exemplo, possui uma estrutura de informações (as classes, atributos, etc.) e uma ou mais representações visuais desta estrutura (representação gráfica, textual, etc.). A expressão modelo se refere à estrutura de informações.

[4] Neste trabalho, a expressão conceito é usada para designar as unidades de informação do domínio de modelagem tratado. No caso da modelagem baseada no paradigma de orientação a objetos, como tratado no ambiente SEA, constituem tipos de conceito, por exemplo, classe, atributo, mensagem, etc.

### 3.1.1 Estrutura do *Framework* OCEAN para a Definição de Documentos

A estrutura de herança das classes abstratas do *framework* OCEAN para suporte à definição de documentos é apresentada na Figura 3.1.

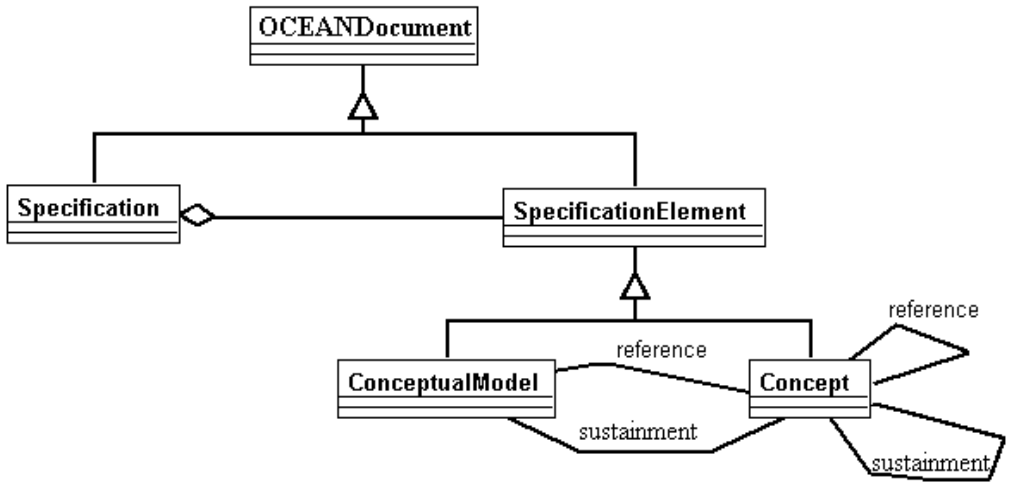



Figura 3.1 – Classes que definem a estrutura de um documento (SILVA, 2000a)

Neste modelo, uma instância de *Specification* (uma especificação) agrega elementos de especificação, ou seja, agrega as instâncias de subclasses de *Concept* e *ConceptualModel*. A classe *Concept* permite a criação de todos os conceitos tratados no ambiente, através de subclasses concretas que modelam cada tipo de conceito tratado. A classe *ConceptualModel* permite a criação dos tipos de modelos do ambiente através de suas subclasses concretas que descrevem os modelos tratados.

A especificação também registra as associações de sustentação[5] e referência[6] entre os pares de elementos de especificação. Através desta hierarquia de herança (todas estas

[5] Associação de sustentação: permite o estabelecimento de ligação semântica entre dois elementos de especificação distintos, em que um elemento assume o papel de elemento sustentador e o outro, de elemento sustentado, onde a permanência de um elemento sustentado em uma especificação depende da permanência do elemento sustentador. Exemplo: uma classe sustenta um método.

classes são abstratas), o *framework* OCEAN suporta diferentes tipos de documentos (conceitos, modelos e especificações). Das classes concretas que serão criadas a partir destas, cria-se uma estrutura específica para cada tipo de documento.

### 3.1.2 Visualização de Elementos de Especificação

Para relacionar cada elemento de especificação das especificações de um ambiente a um mecanismo de visualização do elemento é usado um gerente de referências, *Reference Manager*. *Reference Manager* possibilita o desacoplamento entre elementos de especificação e mecanismos de visualização, possibilitando reuso isolado de cada estrutura e novas associações entre elas. A Figura 3.2 mostra a estrutura de edição de elementos de especificação do *framework* OCEAN.

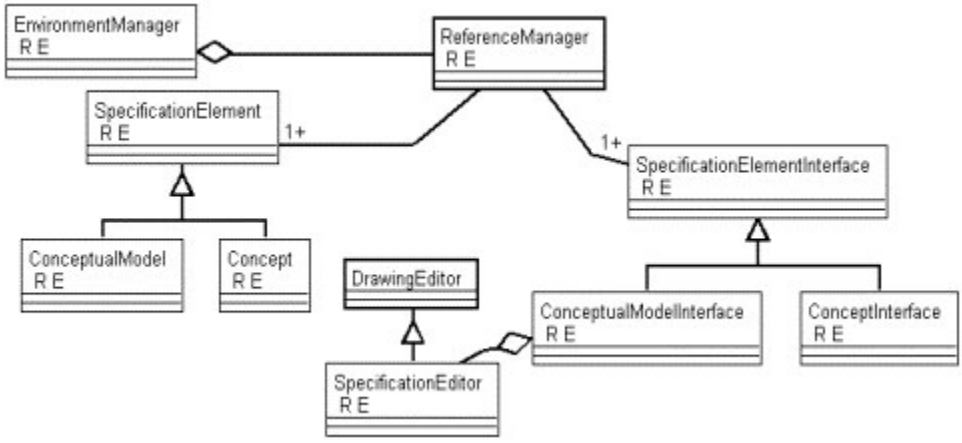



Figura 3.2 – Estrutura de edição de elementos de especificação do OCEAN (SILVA, 2000a)[7]

---

[6] Associação de referência entre elementos de especificação estabelece que parte da definição de um elemento de especificação, o elemento referenciador, depende da referência a outro elemento, o elemento referenciado. Exemplo: uma mensagem referencia um método.

[7] "R" representa a Redefinibilidade de classe, que estabelece no *framework* se as classes podem ou não originar subclasses nos artefatos desenvolvidos sob o *framework*. "E" representa a E ssencialidade de uma classe, que corresponde a registrar que todo artefato de software produzido a partir desse *framework* deve utilizar essa classe. Somente classes redefiníveis podem ser essenciais.

Ao criar subclasses concretas de *SpecificationElementInterface* novos mecanismos de visualização de conceitos e modelos podem ser definidos. Uma instância de *ConceptualModelInterface* agrega um editor gráfico (no *framework* OCEAN, foi usado uma extensão do *framework HotDraw*). A classe *ConceptInterface* guarda a estrutura genérica dos mecanismos de visualização de conceitos e a classe *ConceptModelInterface* guarda a estrutura genérica dos mecanismos de visualização de modelos.

### 3.1.3 Armazenamento de Especificações

O *framework* OCEAN provê flexibilidade na definição do mecanismo de armazenamento de especificações de um ambiente. O protocolo de armazenamento e recuperação de especificações está definido na classe *StorageManager* (Figura 3.3). O protótipo do *framework* OCEAN possui uma subclasse desta classe, *BOSSStorageManager*, que possibilita o armazenamento de especificações em um arquivo. Para definir um outro dispositivo de armazenamento, define-se outra subclasse de *StorageManager*.

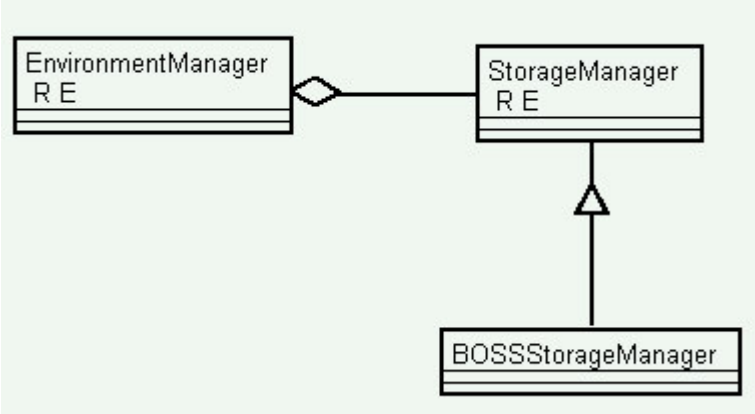



Figura 3.3 – Estrutura de suporte ao armazenamento (SILVA, 2000a)

### 3.1.4 Estruturas de Suporte à Edição Semântica de Especificações

O *framework* OCEAN permite ações de edição sobre a estrutura de informações da especificação. Ao serem efetuadas tais modificações, elas se refletem nas apresentações visuais dos elementos afetados. A ênfase à edição semântica garante o estabelecimento de restrições às ações de edição, impedindo que ações indevidas ocorram.

Na edição de uma especificação, em um ambiente sob o *framework* OCEAN, criam-se conceitos e modelos, removem-se conceitos e modelos, alteram-se conceitos, transferem-se conceitos, fundem-se conceitos, copiam-se e colam-se conceitos. Neste trabalho não são caracterizadas estas ações. O tema é explorado com maior profundidade em (SILVA, 2000a).

No *framework* OCEAN, parte da responsabilidade das ações de edição é delegada aos elementos de especificação. Os elementos de especificação seguem requisitos funcionais na criação de novos tipos de elementos.

### 3.1.5 Estruturas de Suporte à Composição de Ações de Edição

O *framework* OCEAN provê flexibilidade para inclusão de estruturas funcionais, denominadas ferramentas, que reutilizam os procedimentos de edição semântica supridos pelo *framework*. As ferramentas estão associadas a uma ou mais especificações e a um ou mais tipos de modelos. Alguns exemplos de ferramentas desenvolvidas: ferramenta de inserção de padrões de projeto, ferramenta de importação de interfaces de componentes, ferramenta de verificação de consistência e ferramentas de edição, análise e transformação de ações. O conjunto de ferramentas utilizáveis é definido na inicialização de um ambiente.

Todo ambiente específico (definido a partir de uma subclasse de *EnviromentManager*) agrega um gerente de ferramentas (Figura 3.4), que é uma instância da classe *ToolManager*.

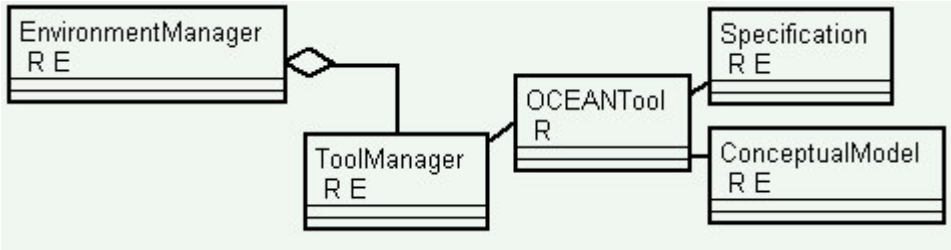



Figura 3.4 – Estrutura de suporte à produção de ferramentas (SILVA, 2000a)

### 3.1.6 Funcionalidades das Ferramentas do *Framework* OCEAN

Assim como os editores, as funcionalidades supridas por ferramentas reutilizam funcionalidades de edição supridas pelo *framework* OCEAN. As ferramentas constituem o meio de inserção de funcionalidades mais flexível dentre as possibilidades oferecidas por OCEAN porque permitem a inclusão de novas ferramentas sem exigir a alteração do restante da estrutura de um ambiente.

As ferramentas podem ser classificadas como: ferramentas de edição (capacidade de ler e alterar uma especificação), ferramentas de análise (capacidade de ler uma especificação sem alterá-la, descrevendo suas características) e ferramentas de transformação (capacidade de importar ou exportar informações, como, por exemplo, produzir código correspondente a uma especificação). A Figura 3.5 mostra os tipos de ferramentas de ambiente do *framework* OCEAN.

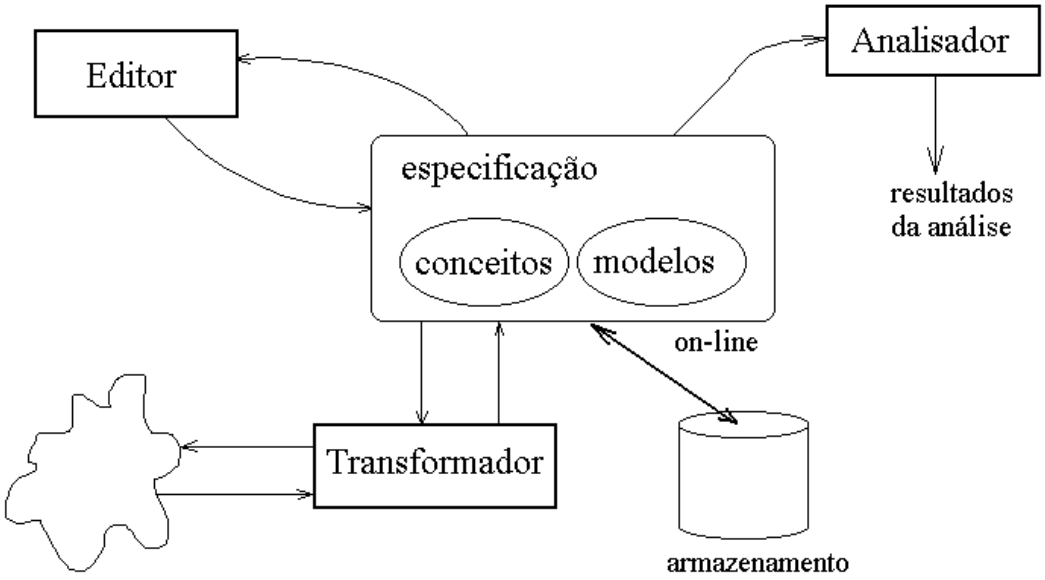



Figura 3.5 – Tipos de ferramentas de ambiente sob o *framework* OCEAN (SILVA, 2000a)

## 3.2 Ambiente de Desenvolvimento de *Software* SEA

O ambiente SEA foi desenvolvido como uma extensão da estrutura de classes do *framework* OCEAN e é voltado ao desenvolvimento e uso de artefatos de *software* reutilizáveis.

SEA provê suporte para o desenvolvimento de *software*, possibilitando a utilização integrada das abordagens de desenvolvimento orientado a componentes e desenvolvimento baseado em *frameworks.* Estas abordagens têm em comum a ênfase em promover reuso, tanto de implementação, como de projeto. SEA suporta o desenvolvimento de *frameworks*, componentes e aplicações como estruturas baseadas no paradigma de orientação a objetos.

O desenvolvimento de *frameworks*, componentes e aplicações em SEA exige a construção das especificações de projeto destes artefatos, que são feitos em UML (com algumas modificações[8]) e depois convertidas em código automaticamente.

A Figura 3.6 apresenta a estrutura de qualquer ambiente desenvolvido sob o *framework* OCEAN, inclusive do ambiente SEA.

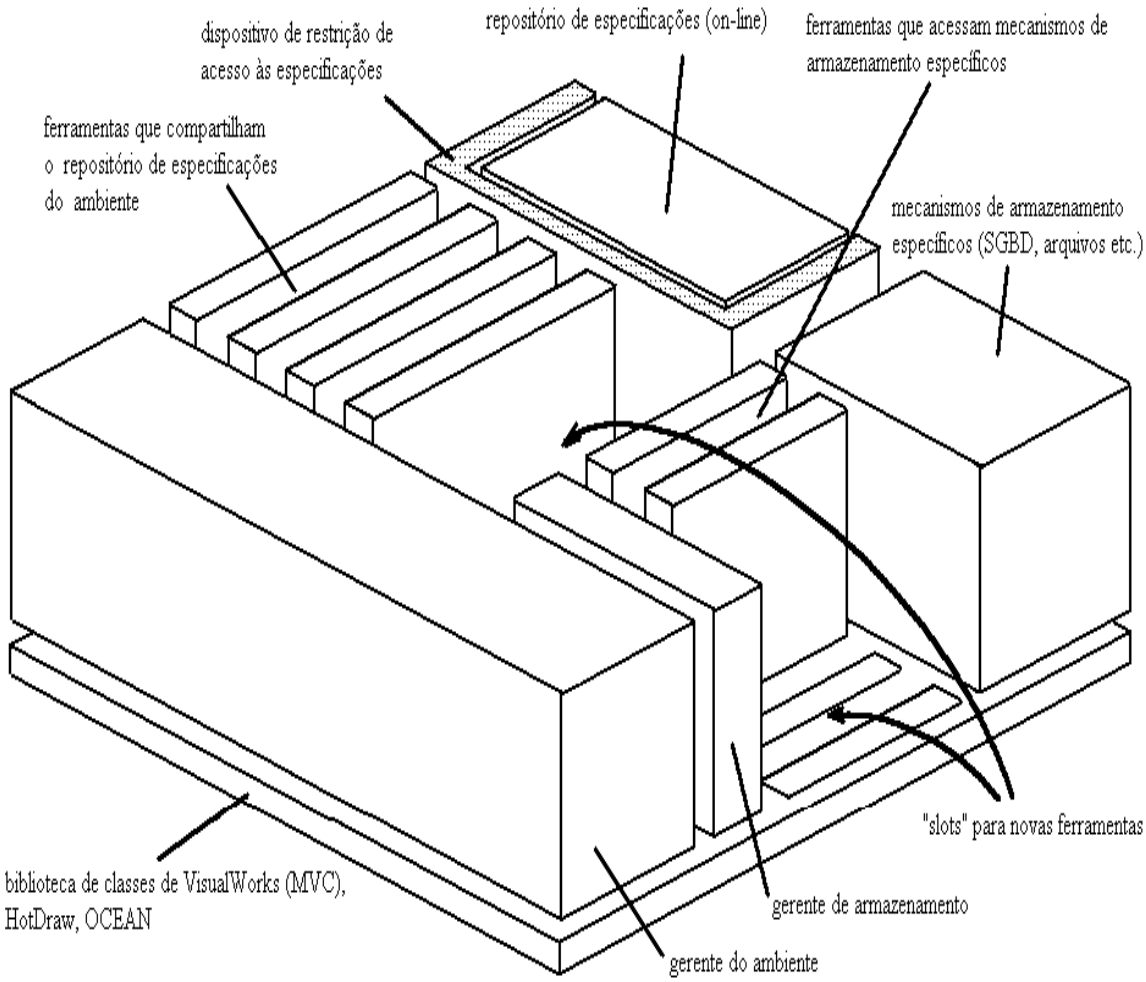



Figura 3.6 – Estrutura do ambiente SEA (SILVA, 2000a)

[8] Por razões de implementação, nem todos os elementos sintáticos previstos de UML foram incorporados aos editores do ambiente SEA, porém, foram introduzidas extensões para: representar conceitos do domínio de *frameworks*, agregar recursos de modelagem a UML, expressar ligação semântica entre elementos de uma especificação, possibilitar que especificações sejam tratadas como hiperdocumentos e descrever o algoritmo dos métodos.

SEA foi desenvolvido para suportar o desenvolvimento e o uso de *frameworks* e componentes, considerando os seguintes requisitos:

- Registro de informação referente à flexibilidade de um *framework* em sua especificação;
- Suporte à produção de artefatos de *software* a partir do uso de um *framework*;
- Suporte à alteração de um *framework* a partir de artefatos produzidos usando este *framework*;
- Suporte à produção e à alteração de *frameworks* a partir de aplicações do domínio tratado;
- Suporte ao desenvolvimento de componentes, considerando o desenvolvimento de interfaces e de estruturas internas de componentes como atividades distintas;
- Suporte à construção de artefatos de *software* a partir da interconexão de componentes;
- Suporte à aplicação conjunta das abordagens de *frameworks* e componentes;
- Suporte ao uso de padrões de projeto em especificações de artefatos de *software*.

### 3.2.1 Especificação no Ambiente SEA

O protótipo do ambiente SEA suporta três tipos de especificação:

- Especificação orientada a objetos (OO);
- Especificação de interface de componente;
- *Cookbook* ativo, para orientar o uso de um *framework* previamente desenvolvido.

As especificações OO no ambiente SEA podem ser construídas para especificar *frameworks*, aplicações ou componentes (flexíveis[9] ou não). Estas especificações podem ser desenvolvidas sob especificações de *frameworks* ou de componentes reusáveis.

O ambiente SEA supre um conjunto de funcionalidades específicas: disponibiliza a criação e edição de modelos (através dos editores gráficos de UML); possibilita a alteração da relação semântica entre os conceitos (através de procedimentos de transferência e fusão); possibilita a criação automática de métodos de acesso a atributos; disponibiliza suporte automatizado à composição do diagrama de corpo de método (a partir da análise dos diagramas de seqüência e dos diagramas de transição de estados); disponibiliza suporte automatizado para a alteração de *frameworks* (com transferência de trechos de especificação); insere de forma semi-automática padrões de projeto e gera código no ambiente SEA.

#### 3.2.1.1 Técnicas de Modelagem do Ambiente SEA para Especificações OO

O ambiente SEA utiliza apenas cinco das oito técnicas de modelagem de UML e acrescenta uma técnica adicional para descrever o algoritmo dos métodos. A seguir, são descritas estas técnicas.

- Diagrama de Classes: os conceitos representáveis no diagrama de classes do ambiente SEA são classe, com seus atributos e métodos, associação binária, agregação e herança. Todos estes conceitos podem ser classificados como externos. Um conceito externo define que um elemento de especificação foi criado fora do contexto da especificação. Uma classe externa denota uma classe referenciada pela especificação, que pode gerar subclasses. Uma restrição estabelecida é que se uma classe for externa, todos os seus atributos e métodos também são externos e se uma classe não for externa, nenhum dos seus atributos e métodos poderão ser externos (exceto os herdados). As classes

[9] Componente flexível: é um componente que possui uma interface e uma especificação com classes redefiníveis. (SILVA, 1999)

externas só podem ter características alteradas através da criação de subclasses.

- Diagrama de Casos de Uso: é composto por casos de uso, atores e relações que interligam atores e casos de uso.

- Diagrama de Atividades: é composto por casos de uso (que correspondem às atividades) e transições entre casos de uso. Existe uma atividade inicial e alguma outra atividade só pode ocorrer depois desta atividade ter iniciado o sistema.

- Diagrama de Transição de Estados: é associado a uma classe e cada estado é definido em termos de atributos da classe e de valores assumidos por estes atributos. Um método da classe modelada é associado a cada transição de estado. No ambiente SEA não são representados estados paralelos e não é possível expandir um estado em sub-estados em um mesmo diagrama (é necessário que seja feito em outro diagrama).

- Diagrama de Seqüência: é composto por mensagens e elementos que trocam mensagens. Uma mensagem é enviada a um objeto, que corresponde a uma instância de uma das classes da especificação, e invoca um método da classe desse objeto. O emissor de uma mensagem pode ser um ator, um objeto ou nenhum emissor. Neste último caso, deve ser necessariamente a primeira mensagem de um diagrama de seqüência e corresponde ao refinamento da execução do método referenciado por esta mensagem. Uma mensagem pode ter um condicionador de ocorrência associado. Em SEA, são previstos os seguintes condicionadores: *if, while, repeat* e *nTimes*.

- Diagrama de Corpo de Método: este diagrama prevê comandos (conversíveis para comandos equivalentes em linguagens de programação) para a descrição dos algoritmos dos métodos. São eles: *Variables, Assignment, Return, If, IfElse, While, Repeat, nTimes, Message, Task e Comment.* Esses comandos permitem que sejam descritos os métodos usados nos diagramas. Os comandos *ExternalStateDiagram e ExternalSequenceDiagram* são usados no

processo de definição do corpo dos métodos, onde a ferramenta de geração de comandos *external* permite que seja feita a varredura dos Diagramas de Seqüência e dos Diagramas de Transição de Estados e produza automaticamente os comandos no Diagrama de Corpo de Método, para auxiliar a definição do algoritmo.

#### 3.2.1.2 Mecanismos de Interligação de Elementos de Especificação

O *framework* OCEAN dispõe de três mecanismos para estabelecer ligações semânticas entre elementos de especificação que são usados no ambiente SEA: *link*s, associações de sustentação e associações de referência. As associações de sustentação e referência foram descritas no item 3.1.2. O registro entre associações de sustentação e referência entre modelos de especificação é feito através de tabelas.

*Link*s: um *link* corresponde a um tipo de conceito (subclasse de *Concept*) que pode ser associado a qualquer elemento da especificação. Os *link*s têm duas finalidades nos ambientes produzidos sob o *framework* OCEAN: possibilitar a criação de caminhos de navegação nas especificações produzidas e estabelecer ligações semânticas entre elementos de uma especificação (mantendo o baixo acoplamento entre as classes envolvidas). Um *link* estabelece uma associação que não afeta diretamente a estrutura dos elementos envolvidos. Exemplo: um diagrama de casos de uso pode apontar um diagrama de seqüência que o refina.

#### 3.2.1.3 Estrutura Semântica de uma Especificação OO

Uma estrutura de especificação é definida a partir de um conjunto de tipos de modelo, de um conjunto de tipos de conceito e de uma rede de associações entre os elementos destes conjuntos. O *framework* OCEAN proporciona aos ambientes criados sob ele, uma estrutura de repositório de conceitos (*ConceptRepository*) que armazena todos os conceitos de uma especificação (as instâncias de subclasses de *Concept*). Este repositório é composto por um conjunto de repositórios, um para cada tipo de conceito tratado pela especificação. A Figura 3.7 esclarece esta estrutura.

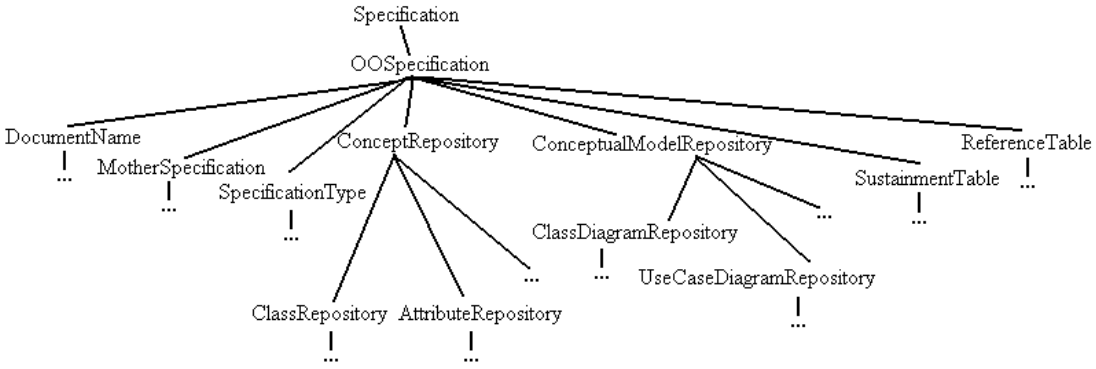



Figura 3.7 – Parte da árvore de derivação de uma especificação OO (SILVA, 2000a)

A estrutura também possui um repositório de modelos (*ConceptualModelRepository*) que contém um repositório para cada tipo de modelo tratado pela especificação.

Nesta estrutura foi adotado o armazenamento centralizado de conceitos. Com isto, todos os conceitos (instâncias) produzidos na construção de modelos são armazenados no repositório de conceitos, podendo ser referenciados por um ou mais modelos. Assim, uma classe presente em mais de um diagrama de classes e que seja referenciada como a classe relacionada a um objeto de um diagrama de seqüência, é um único elemento da especificação. Qualquer alteração sobre esta classe produz efeito sobre todos os elementos de especificação que a referenciam.

Um tipo de conceito é definido como uma estrutura de informações, que podem corresponder a referências a outros conceitos ou informações exclusivas do conceito tratado. Tipos de modelos são caracterizados pelos tipos de conceitos referenciados. Modelos mantêm referências a conceitos que definem sua estrutura.

### 3.2.2 Desenvolvimento Baseado em Componentes no Ambiente SEA

Componentes são unidades de composição que possuem interface especificada que permite a conexão com um ou mais componentes. Os componentes podem ser

independentes de linguagem, plataforma e localização, o que possibilita que componentes escritos em diferentes linguagens, ou componentes fisicamente distantes ou ainda componentes elaborados em diferentes plataformas, sejam interligados entre si.

O desenvolvimento de aplicações utilizando componentes consiste em adquirir ou desenvolver componentes de interesse, para amarrá-los com outros componentes e, assim, formar uma aplicação (KLABUNDE, 2000).

O uso de componentes tem como objetivo proporcionar o desenvolvimento de aplicações simplesmente a partir da montagem de componentes já existentes e ainda possibilitar a substituição de objetos em tempo de execução (KLABUNDE, 2000).

O desenvolvimento de *software* baseado em componentes traz alguns benefícios, entre eles: a reutilização de artefatos de *software* de alta granularidade (já depurados, minimizando erros), a delegação e divisão de parte das responsabilidades e redução do esforço no desenvolvimento de *software*s.

O ambiente SEA proporciona o desenvolvimento e uso de componentes. O desenvolvimento de componentes em SEA é definido através de especificações orientadas a objetos. O que caracteriza a especificação de um componente é que parte da especificação corresponde à descrição de uma interface, isto é, um conjunto de classes responsável pela comunicação do componente com o meio externo.

O uso dos componentes em SEA ocorre quando componentes contidos na biblioteca de especificações do ambiente são importados para especificações orientadas a objetos. A inserção de componentes é feita pela ferramenta de importação de componentes do ambiente, que insere apenas a classe fachada do componente.

### 3.2.3 *Cookbook* Ativo – Suporte ao Uso de *Frameworks* no Ambiente SEA

*Cookbooks* ativos dão suporte ao uso de *frameworks* no ambiente SEA através de seus documentos manuseáveis que dão instruções ao usuário de como usar um *framework*.

*Cookbooks* são hiperdocumentos que possuem *link*s ativos permitindo ações de edição automatizadas, entre elas: criação de especificação sob o *framework* descrito, criação de elementos de especificação e avaliação do cumprimento das restrições impostas no projeto do *framework*. Porém, são incapazes de descrever todas as possibilidades de criação de artefatos de *software* a partir de um *framework*.

No ambiente SEA, o uso de *cookbooks* ativos serviu para estabelecer alguns requisitos no mecanismo de suporte ao uso de *frameworks*: direcionar as ações dos usuários de *frameworks*, reduzir o esforço para entender o projeto do *framework*, liberar o usuário de atividades de baixo nível e verificar a satisfação dos requisitos para a geração de aplicações a partir de um *framework*.

### 3.2.4 O Ambiente SEA e o Gerenciamento do Processo do Desenvolvimento

O ambiente SEA suporta a criação de especificações de diferentes tipos, supre as características de *framework*s, permitindo a ligação semântica entre elementos de especificação e entre especificações distintas (o que não é comum aos ambientes de desenvolvimento de *software*), mas não dá suporte ao gerenciamento durante o desenvolvimento das aplicações.

Pela característica de flexibilidade do ambiente SEA, é possível inserir gerenciamento à sua estrutura. Com a inclusão do gerenciamento, um novo ambiente de desenvolvimento de *software* é constituído, e é chamado SEA2. O novo ambiente suporta todo o aspecto tecnológico previamente existente e agrega o suporte ao gerenciamento de processos de desenvolvimento de *software*.

O gerenciamento pode ser feito através da implementação de *Workflow* e da adaptação de requisitos de CMM à nova estrutura do sistema gerenciador dos processos.

# 4 MODELO DE MATURIDADE DE CAPACITAÇÃO

CMM (*Capabiliy Maturity Model*) é um modelo de gerenciamento de processos que busca qualidade na produção de *software*. (PAULK et. al., 1994)

CMM insere práticas aplicáveis a processos produtivos no desenvolvimento de um *software*. Como o ambiente SEA não suporta gerenciamento nem qualidade no desenvolvimento de suas aplicações, CMM é adotado para suprir esta lacuna.

CMM estabelece requisitos para planejar, organizar, acompanhar e supervisionar o desenvolvimento do *software* e *Workflow*, através da sua modelagem, aplica os requisitos propostos por CMM.

Dos níveis existentes em CMM, o presente capítulo dá ênfase ao nível 2 e suas respectivas KPAs, pois algumas destas KPAs são implantadas no ambiente SEA2.

## 4.1 Histórico

CMM surgiu na Universidade *Carnegie Mellon*, nos Estados Unidos, no SEI (*Software Engineering Institute*) por volta de 1986, com o objetivo de ajudar as organizações a aperfeiçoarem os processos de desenvolvimento de um *software*. Este esforço foi iniciado devido a uma solicitação do Governo Federal Norte-Americano (Departamento de Defesa dos Estados Unidos) que necessitava avaliar a capacidade de produção de *software* de empresas contratadas.

Após anos de estudos e experiências com a maturidade dos processos de *software*, o caminho de evolução de processos foi definido em cinco níveis de maturidade: Inicial, Repetível, Definido, Gerenciado e Otimizado, que são usados atualmente. (PAULK et. al., 1994)

Nestes níveis foram introduzidas práticas aplicáveis a processos produtivos ao desenvolvimento de *software*. Dentre elas, planejamento, engenharia para controle dos processos de *software*, gerenciamento, manutenção e priorização de processos, gerando

assim melhoria da qualidade, da funcionalidade e do custo do próprio *software*. (PAULK et. al., 1994)

A primeira versão do livro de CMM saiu em agosto de 1991 (versão 1.0) e o seu aprimoramento em março de 1994 (versão 1.1). Porém, os testes de avaliação e aperfeiçoamento estão continuamente sendo feitos e CMM vai evoluir de acordo com os retornos de seus usuários e de acordo com as mudanças nos padrões e modelos futuros. (PAULK et. al., 1994)

## 4.2 Características

CMM consiste de um conjunto de estágios e metas a serem alcançadas por uma organização para conseguir um aperfeiçoamento no desenvolvimento de seus *softwares*. A empresa que segue todos os requisitos de CMM garante maturidade e capacidade de evolução do seu processo de *software*. (PAULK et. al., 1994)

As metas de CMM preocupam-se com o processo em um ambiente de desenvolvimento de *software*, mais especificamente sob o ponto de vista gerencial. CMM não trata do aspecto tecnológico nem mesmo do produto. Sua ênfase é a busca de qualidade através do gerenciamento dos processos. A figura abaixo apresenta onde o modelo CMM se encaixa dentro de uma perspectiva de qualidade.

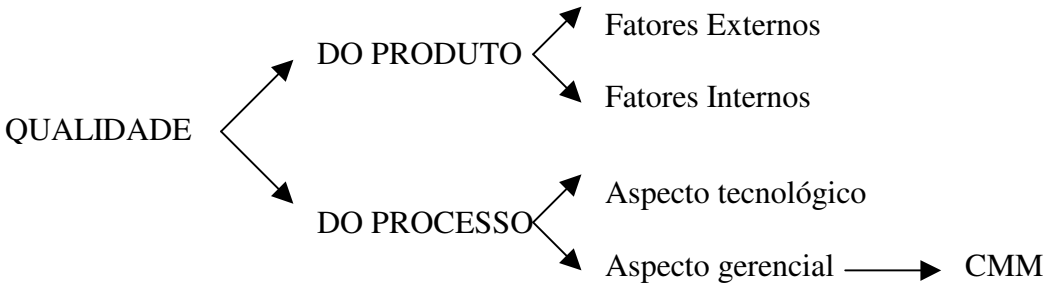



Figura 4.1 – CMM sob o ponto de vista de qualidade

O modelo CMM propõe que seja feito um trabalho voltado à estrutura organizacional do desenvolvimento de *software*, tendo como meta maturidade no

processo de desenvolvimento. Para CMM, a qualidade do processo torna-se visível no momento em que o processo torna-se maduro. Entretanto, a maturidade leva tempo e ocorre de maneira gradual.

Cada nível de maturidade de CMM tem os seus próprios requisitos e eles são cumulativos, ou seja, não se pode passar para um próximo nível sem ter todos os requisitos deste nível cumpridos. A cada nível alcançado, a empresa pode ser certificada por esta conquista.

A figura 4.2 mostra a seqüência dos níveis de CMM.

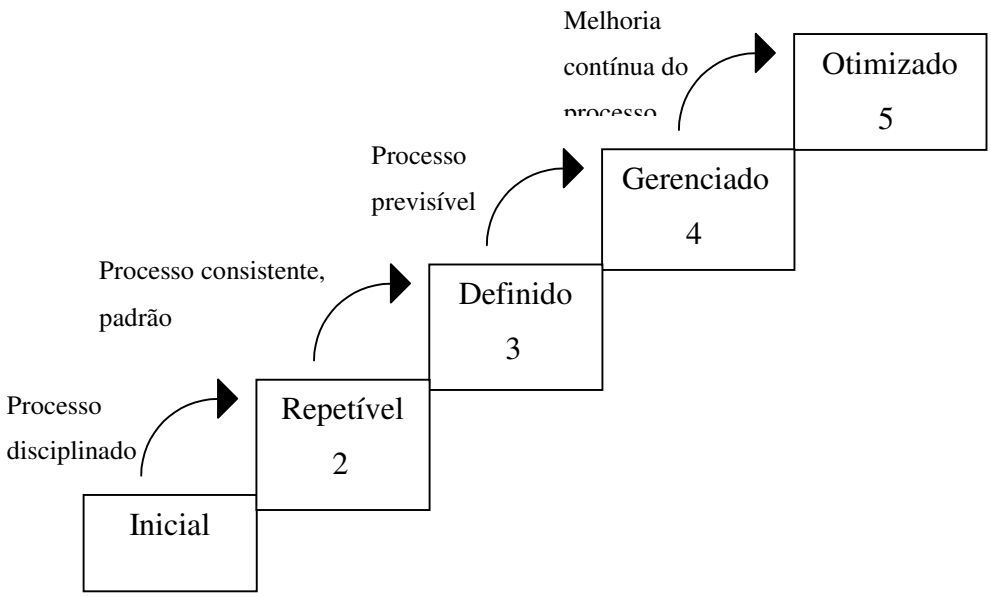



Figura 4.2 – Os cinco níveis de CMM (PAULK et. al., 1994)

Cada nível de CMM é caracterizado por um conjunto de "áreas de processo", mais conhecidas como KPAs (*process areas*). As KPAs são agrupadas por características comuns e contêm as práticas-chave. As práticas-chave são atividades e infraestrutura que contribuem para a implementação e institucionalização da KPA (PAULK et. al., 1994). A Figura 4.2 mostra esta estrutura.

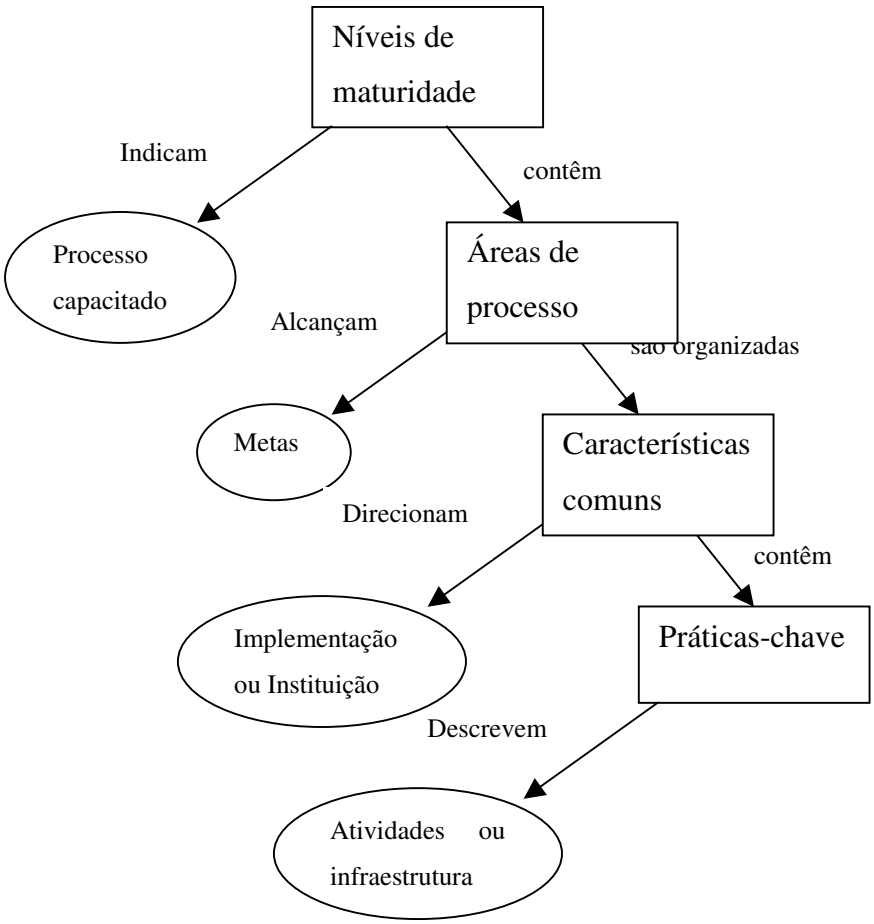



Figura 4.3 – A estrutura de CMM (PAULK et. al., 1994)

## 4.3 Descrição dos Níveis de CMM

Nível 1: neste nível poucos processos estão definidos e o sucesso depende individualmente do esforço do desenvolvedor. O *software* é desenvolvido sem os requisitos de gerenciamento e não há controle do processo. Como não existem requisitos gerenciais para estar no nível 1, qualquer um que tenha capacidade de desenvolver *software* está incluído neste nível. O nível 1 não possui KPAs.

Nível 2: este nível estabelece o gerenciamento de processos, controla as estimativas, planejamento e compromissos estabelecidos. Os problemas são reconhecidos e corrigidos na medida que ocorrem. As suas KPAs são as seguintes: gerenciamento de requisitos; planejamento do projeto de *software*; acompanhamento e supervisão do

projeto de *software*; gerenciamento de subcontrato de *software*; garantia de qualidade de *software* e gerenciamento de configuração de *software*.

Nível 3: o processo para gerenciamento e coordenação das atividades é documentado, padronizado e integrado ao modelo de processo de *software* da organização. É definido um processo global para a organização e cada projeto da organização adapta este processo às suas características. As KPAs do nível 3 são as seguintes: foco no processo da organização; definição do processo da organização; programa de treinamento; gerenciamento de *software* integrado; coordenação intergrupos e revisões.

Nível 4: as estimativas do processo de *software* e da qualidade do produto são coletadas, interpretadas e controladas. Como a organização pode interpretar os números obtidos no seu desenvolvimento, pode tomar decisões (de planejamento e gerência) baseadas nesses números. Os desvios dos objetivos são verificados ao longo do processo e ações corretivas podem ser tomadas preventivamente. São identificadas também as tendências na qualidade do processo e do produto e, portanto, limites são estabelecidos. KPAs do nível 4: gerenciamento quantitativo do processo e gerenciamento de qualidade de *software*.

Nível 5: como a organização consegue avaliar seus processos, ela é capaz de atacar seus pontos fracos e eliminar os problemas identificados. Assim, aumenta a qualidade e otimização dos processos. Continuamente os processos são melhorados também através de novos procedimentos, novas ferramentas e novas tecnologias. As KPAs do nível 5 são as seguintes: prevenção de defeitos; gerenciamento de mudança de tecnologia e gerenciamento de mudança do processo.

## 4.4 Caracterizando uma KPA

Uma KPA é um requisito a ser cumprido em CMM (PAULK et. al., 1994). Sua estrutura é a seguinte:

- Texto introdutório;
- Conjunto de metas;

- Conjunto de práticas-chave, onde cada prática-chave pode estar entre uma das seguintes classificações: compromissos para realizar o cumprimento do requisito, aptidões para realizar o cumprimento do requisito, atividades realizadas, medidas e análise, verificações de implementação.

## 4.5 Descrição das KPAs do Nível 2

Este trabalho enfatiza o nível dois de CMM, pois este nível possui requisitos que possibilitam gerenciar um projeto. Este projeto pode ter documentado e controlado suas estimativas, planejamento e compromissos estabelecidos. Os problemas que aparecem no decorrer do desenvolvimento do projeto são reconhecidos e corrigidos. A seguir, as KPAs do nível dois são detalhadas de acordo com (PAULK et. al., 1994).

### 4.5.1 KPA Gerenciamento de Requisitos

Esta KPA propõe estabelecer e manter uma concordância com o cliente sobre os requisitos do projeto de *software*. O projeto segue uma política organizacional escrita para gerenciar os requisitos, onde os planos de *software*, produtos e atividades (que são revisadas periodicamente) são mantidos consistentes com os requisitos.

Para cada projeto é estabelecida a responsabilidade de analisar os requisitos do sistema, alocando-os para hardware, *software* e outros elementos do sistema, sempre documentando o que for alocado. No caso de mudança dos requisitos alocados, os novos requisitos são analisados e incorporados ao projeto de *software*. Existem membros que são treinados para realizar suas atividades de gerenciamento de requisitos, sendo que recursos e fundos são destinados a isso.

Medições são feitas e usadas para determinar o estado das atividades de gerenciamento de requisitos alocados e, através do grupo de garantia de qualidade, o *software* é revisado. Esta revisão consiste de auditoria nas atividades e produtos de trabalho do gerenciamento de requisitos e reporta os resultados.

### 4.5.2 KPA Planejamento do Projeto de *Software*

O planejamento do projeto de *software* estabelece planos de desenvolvimento e gerenciamento para o projeto de *software*. Nesta etapa são definidos o planejamento de todo o projeto (e seu ciclo de vida), os compromissos que devem ser cumpridos (e por quem) e as estimativas do trabalho a ser realizado.

O planejamento do projeto de *software* preocupa-se com as etapas do desenvolvimento, com os produtos a serem gerados, com as estimativas (esforço, prazo e custo), com as ferramentas necessárias (ferramentas de desenvolvimento, gerenciamento e comunicação) e com o levantamento dos riscos e alternativas para solucioná-los.

O planejamento segue uma política organizacional escrita, onde os documentos estabelecem o trabalho a ser feito. De acordo com cada produto de trabalho, é elaborada uma estimativa do "tamanho" do produto d e *software* que é produzido e, com este dado são produzidos estimativas de esforço, custo e recurso computacional.

Com o cronograma do projeto estabelecido, os riscos são identificados, avaliados e documentados. Periodicamente (ou motivadas por evento), revisões são feitas no planejamento do projeto, buscando melhorias nas atividades e produtos.

### 4.5.3 KPA Acompanhamento e Supervisão do Projeto de *Software*

A KPA Acompanhamento e Supervisão do Projeto de *software* está associada ao desenvolvimento do projeto de *software*, controlando o processo de acordo com o planejamento definido.

No acompanhamento e supervisão do projeto de *software* os resultados e performance são acompanhados (de acordo com o planejamento) e, se algum resultado desvia-se do plano, ações corretivas são executadas. As alterações corretivas são sempre aprovadas pelo gerente e comunicadas ao grupo de Engenharia de *Software*.

O tamanho dos produtos, custos, esforço estimados, atividades realizadas também são acompanhados e ações corretivas são efetuadas, se necessário. Os recursos computacionais críticos, cronogramas e riscos são acompanhados e revisões periódicas e formais são feitas.

### 4.5.4 KPA Gerenciamento de Subcontrato de *Software*

O gerenciamento de *Sobcontrato* de *software* ocorre quando uma organização é (sub) contratada para desenvolver parte do projeto de *software*. Suas metas são selecionar subcontratados qualificados, estabelecer com o subcontratado concordância em relação aos compromissos de cada parte e ter uma comunicação permanente com o subcontratado.

O projeto segue uma política organizacional escrita para o gerenciamento do subcontrato de *software*. Existe um gerente de subcontrato que é designado a ser responsável pela verificação e gerenciamento do subcontrato de *software*. Este gerente deve ser treinado para realizar estas atividades.

O trabalho a ser subcontratado é definido e planejado de acordo com um documento revisado e aprovado pelo contratante. Periodicamente revisões e mudanças podem ser efetuadas no subcontrato de *software*.

### 4.5.5 KPA Garantia de Qualidade de *Software*

O propósito da KPA garantia de qualidade de *software* é prover gerenciamento dos processos que estão sendo usados pelo projeto de *software* e pelos produtos que estão sendo construídos.

Garantia de qualidade de *software* envolve revisão e auditoria dos produtos e atividades do *software* para verificar se eles seguem os procedimentos estabelecidos. No caso de alguma falha ou lacuna, membros do grupo de SQA executam as atividades necessárias para a resolução dos problemas.

As práticas-chave identificam as atividades específicas da garantia de qualidade de *software* e recursos adequados e previstos estão disponíveis para a execução dessas atividades.

Dentre as principais atividades da SQA, destacam-se: planejamento (início) e acompanhamento do projeto, elaboração e seguimento de um plano, verificação da conformidade entre as atividades, auditoria de produtos e retorno de resultados para o grupo de Engenharia de *Software*.

### 4.5.6 Gerenciamento de Configuração de *Software*

A KPA gerenciamento de configuração de *software* procura estabelecer e manter a integridade dos produtos do projeto de *software* ao longo do ciclo de vida. Isto inclui o controle de versões (que partes constituem cada versão do *software* e o armazenamento de todas as versões de cada parte) e o controle de mudanças.

As atividades desta KPA incluem a elaboração de um plano a ser seguido, manutenção de uma biblioteca de gerenciamento de configuração, solicitação de mudanças e notificações de problemas, criação de produtos, produção de relatórios de atividades e auditoria dos repositórios.

## 4.6 CMM no Ambiente SEA

Das KPAs do nível dois, três delas são de maior importância para este trabalho: Planejamento de Projeto de *Software,* Acompanhamento e Supervisão do Projeto de *Software* e Garantia de Qualidade de *Software*. Estas áreas de processo devem ter seus requisitos cumpridos na implementação de um processo do nível 2 de CMM.

Essas KPAs podem ser tratadas por *Workflow* no ambiente SEA, buscando proporcionar qualidade ao desenvolvimento de *software*. As demais KPAs do nível 2 estão fora do escopo do presente trabalho, pois demandam soluções que não podem ser tratadas por *Workflow*.

A KPA Planejamento de Projeto de *Software* pode ser associada à ferramenta de modelagem de processos de *Workflow*. Nesta etapa, são estabelecidos os responsáveis por cada processo, a ordem a ser seguida pelos processos e o que cada processo deve resultar.

A KPA Acompanhamento e Supervisão do Projeto de *Software* produz relação com *Workflow* através do possível acompanhamento do desenvolvimento do projeto, permitindo a visualização de processos executados, processos a serem executados e processos que aguardam permissão para serem cumpridos. Com o auxílio desta KPA, ações corretivas podem ser realizadas caso algum processo esteja incompatível com o que foi estabelecido no Planejamento.

A KPA Garantia de Qualidade de *Software* auxilia no momento em que as políticas escritas (Planejamento) são comparadas com o que realmente foi executado durante o projeto. Podem ser efetuadas revisões e auditorias em relação aos processos cumpridos.

Os próximos capítulos apresentarão a tecnologia de *Workflow* e a possível união de CMM com esta tecnologia visando o gerenciamento de processos do ambiente de desenvolvimento de *software* SEA.

# 5 *WORKFLOW*

Este capítulo apresenta características, conceitos, funcionalidades, vantagens e apresentação do metamodelo da tecnologia de *Workflow*. Esta descrição faz-se necessária para um melhor entendimento do trabalho proposto, que é o de implementar sistemas de *Workflow* em um ambiente de desenvolvimento de *software*.

## 5.1 Por que *Workflow* no ambiente SEA

Por *Workflow* entende-se a automatização de um processo de negócio, durante a qual, documentos, informações e/ou atividades são passadas de um participante a outro, a fim de que sejam tomadas ações, de acordo com um conjunto de regras e procedimentos (*WORKFLOW MANAGEMENT COALITION*, 1999).

A tecnologia de *Workflow* foi escolhida para este trabalho por ser indicada para modelar e controlar totalmente as atividades que devem ser executadas em um sistema baseado em processos, o caso do ambiente SEA. Outras opções poderiam ser usadas, (por exemplo, Redes de Petri), mas optou-se por dar continuidade ao trabalho já pesquisado por (SCHEIDT, THOM, 1999).

O uso de *Workflow* garante que todas as informações (ou documentos) produzidas em uma etapa sejam enviadas como dados de entrada para a atividade seguinte, sucessivamente até o final do processo (AMARAL, 1997).

Algumas fases do desenvolvimento podem ocorrer simultaneamente (ou não), com a atuação de vários desenvolvedores e várias atividades, dependendo de decisões e regras para serem ativadas. Com *Workflow*, tem-se o conhecimento dos usuários que estão executando cada uma das atividades, em tempo de execução, e tem-se uma visão das informações que estas atividades manipulam em um determinado momento. O ambiente SEA2 precisa desse controle de atividades e decisões decorrentes dos processos a serem desenvolvidos.

O gerenciamento do *Workflow* revisa se os quesitos propostos no projeto foram implementados de acordo com certas regras e inclui seus próprios meios de controle de execução de tarefas. No desenvolvimento de um *software*, o gerenciamento é fundamental para administrar as atividades, controlar o fluxo dos processos e proporcionar qualidade ao produto final. A coordenação do projeto é feita continuamente possibilitando que gerentes/usuários tenham uma idéia exata de todas as características dos processos em termos quantitativos.

Este projeto desenvolveu uma ferramenta para resolver os problemas de gerenciamento de processos e usou *Workflow*, para poder interferir diretamente nos processos, estabelecer regras ao desenvolvimento, alterar processos e gerenciar o desenvolvimento de *software*s.

## 5.2 Histórico

Os primeiros produtos de *Workflow* surgiram no final dos anos 80, sendo utilizados, basicamente, para roteamento de imagens e documentos eletrônicos. A motivação para o uso do *Workflow* restringia-se a redução no tempo de execução dos processos, fato que ocasionava grande impacto nos custos da organização (SETTI, 2000). O maior avanço da tecnologia ocorreu no início da década de 90, quando a mesma passou a ser valorizada, principalmente, pela grande eficiência que apresentava na automatização e controle do fluxo de atividades dos processos de negócio (NICOLAU, 1996).

Os objetivos de *Workflow* são de simplificar, agilizar e dar maior segurança aos processos de negócios, realizando uma divisão mais eficaz do trabalho e melhorando o processo de tomada de decisão.

Por causa da maior necessidade de documentação, padronização e coordenação dos processos de negócio *Workflow* está sendo adotado em várias organizações que vêem como atrativo as suas principais funcionalidades: roteamento de trabalhos; invocação automática de aplicativos; distribuição dinâmica do trabalho; priorização de trabalhos; redução de custos, tempo, erros e redundância na execução dos processos; o

maior controle e qualidade na execução dos processos; a possibilidade de integração de tecnologias já existentes na organização e a integração com a *World Wide Web*. (AMARAL, 1997)

Em 1993 foi criada uma entidade, denominada *Workflow Managment COALITION* (WfMC), que tem a missão de promover a área de *Workflow* através da divulgação da tecnologia e do desenvolvimento de padrões para a interoperabilidade de sistemas de *Workflow*, tanto entre si quanto com outros sistemas de informação (THOM, 2000). Este órgão elaborou um glossário, o qual define um vocabulário de termos para a área de *Workflow* (*WORKFLOW MANAGEMENT COALITION*, 1999).

## 5.3 Conceitos Associados a *Workflow*

Nesta Seção são definidos alguns conceitos básicos relacionados com *Workflow*, que permitem uma completa compreensão desta tecnologia. A referência bibliográfica para esta Seção é (*WORKFLOW MANAGEMENT COALITION*, 1999).

### 5.3.1 Processo de Negócio

Um processo de negócio compreende um conjunto de um ou mais procedimentos ou atividades estruturadas, as quais, coletivamente, realizam um objetivo de negócio no contexto de uma estrutura organizacional.

Um processo de negócio consiste de uma rede de atividades e seus relacionamentos, critérios de início e término do processo, e informações sobre as atividades em si, como participantes, aplicativos e dados de sistema de informação relacionados.

O ciclo de vida de um processo de negócio pode ter minutos, dias ou meses, dependendo da sua complexidade e da duração das várias atividades constituintes.

### 5.3.2 Instância de Processo

Instância de processo é uma representação de um único processo. Ao ser iniciado um processo, a sua instância também é criada. O responsável pela criação,

gerenciamento e terminação de uma instância de processo é o Sistema de gerenciamento do *Workflow*.

### 5.3.3 Sistema de Gerência de *Workflow*

Um Sistema de Gerência de *Workflow* (WFMS) define, cria e gerencia a execução de processos de *Workflow*. Ele é capaz de interpretar a definição do processo de negócio, interagir com os participantes e, quando necessário, invocar ferramentas e aplicações de sistemas de informação.

O WFMS é responsável por controlar o andamento do processo, seguindo rigorosamente a ordem definida pelas dependências entre as atividades, e garantindo as regras de consistência especificadas. Caso o usuário consulte o estado atual de um *Workflow*, o sistema gerenciador reporta o estado corrente da aplicação (AMARAL, 1997).

Os principais WFMS disponíveis, atualmente, no mercado, são *Oracle Workflow* e *Domino Workflow* desenvolvidos, respectivamente, pelas Empresas *Oracle* (ORACLE, 2002) e *Lótus* (LOTUS, 2002).

As literaturas referem-se ao *Oracle Workflow* como um sistema gerenciador de sistemas de *Workflow* que facilita a reengenharia dos processos de negócios. Dentre as principais atividades possibilitadas por este, estão: o roteamento de informações, a automação de processos de negócios e a definição de regras de negócios que podem ser alteradas.

O *Oracle Workflow* permite criar ou modificar processos de negócios; definir os componentes de um processo de *Workflow*; utilizar comandos *SQL* para satisfazer particularidades de um processo de negócios e alterar o estado das atividades. (ORACLE, 2002)

O gerenciamento do *Workflow* é feito através de um conjunto de tabelas e *procedures* (Java) que gerenciam a criação e execução do processo de *Workflow*. Este gerenciamento do *Workflow* acessa a base de dados do *Oracle* ao invés do uso de redes, o que reduz o tráfego de rede. Usando chamadas *PL/SQL*, uma aplicação está apta a

notificar ao *Oracle Workflow* que uma atividade pré-definida está completada. (ORACLE, 2002)

O gerenciador *Domino* (LOTUS, 2002), pertencente ao pacote *Lotus Notes*, introduz gerenciamento baseado em políticas que ajudam a manter um conjunto padrão de configurações básicas a cada novo *Workflow* . No Administrador do Domino pode-se saber os estados particulares de processos, pode-se parar o andamento dos processos, bloqueando os *trigger*s, pode-se solicitar informações dos processos e saber as estatísticas dos trabalhos em andamento. Através do envio de mensagens, as informações são roteadas e armazenadas entre os participantes do *Workflow*, permitindo o controle dos processos e os estados das atividades.

### 5.3.4 Atividade

Atividade é uma descrição de um fragmento de trabalho que forma um passo lógico dentro de um processo. Pode-se dizer, também, que uma atividade é um conjunto de eventos que ocorrem sob a responsabilidade de um ator.

Uma atividade é a menor unidade de trabalho em um processo de negócio, mesmo que possa resultar em vários itens de trabalho que serão atribuídos a um participante do *Workflow*.

A atividade pode ser manual, que não suporta automação computacional, ou automatizada. Uma atividade automatizada requer recursos humanos e/ou de máquina para suportar a execução do processo; onde um recurso humano é requerido, uma atividade é alocada para um participante do *Workflow*.

### 5.3.5 Atividade Manual

É uma atividade dentro de um processo de negócio que não é capaz de ser automatizada e, portanto, está fora do escopo de um sistema gerenciador de *Workflow*. Cada atividade pode ser incluída dentro de um processo, mas não faz parte de um resultado do *Workflow*. São considerados atividades manuais, os trabalhos humanos e os trabalhos manuais.

### 5.3.6 Atividade Automatizada

Atividade que é capaz de ser automatizada e que usa o sistema gerenciador de *Workflow* para gerenciá-la durante a execução do processo de negócio do qual ela faz parte.

Durante a execução de um processo, a atividade automatizada é gerenciada pelo sistema gerenciador de *Workflow*, que permite: a invocação direta de uma aplicação; a capacidade de um ou mais trabalhos serem assinados por um participante, com suporte a ferramentas e aplicações também invocadas pelo sistema gerenciador de *Workflow* e, a capacidade de uma ou mais tarefas sejam assinadas por um participante do *Workflow* (para processamento independente do sistema gerenciador de *Workflow*), com o participante notificando a conclusão dos itens de trabalho ao sistema gerenciador de *Workflow*.

### 5.3.7 Instância de Atividade

É a representação de uma atividade dentro de uma instância de processo. Cada instância de atividade representa uma invocação única de uma atividade, relacionada exatamente com uma instância de processo.

Uma instância de atividade é criada e gerenciada pelo Sistema de gerenciamento do *Workflow*.

### 5.3.8 Participante

O participante do *Workflow* é um recurso que executa o trabalho representado por uma instância de atividade de um *Workflow* (é aquele que atua). Ele é o responsável pela execução de uma ou mais atividades pertencentes ao *Workflow* e pode ser tanto um ser humano, quanto um *software*. Um participante, também pode ser chamado de ator, agente, usuário ou executor do trabalho.

### 5.3.9 Papel

Quando um conjunto de participantes possui mesmas características e qualificações, pode-se definir um papel para representar este grupo. Dessa forma, ao

definir-se um *Workflow*, ao invés de associar um participante (nome da pessoa) à atividade, associa-se um papel. Um mesmo participante pode executar mais de um papel.

### 5.3.10 Papel no Processo

Um conjunto de atividades de um processo de *Workflow* que são assumidas e executadas por um participante no *Workflow* com o propósito de satisfazer os objetivos do processo.

### 5.3.11 Item de Trabalho

Para o participante do sistema de *Workflow*, as atividades são apresentadas como uma coleção de itens de trabalho. Cada item de trabalho é o resultado da instanciação de uma atividade. Esta atividade pode possuir documentos e aplicações associadas a ela. Conforme (*WORKFLOW MANAGEMENT COALITION*, 1999), os termos objeto de trabalho, elemento e tarefa são sinônimos, do termo item de trabalho.

### 5.3.12 Lista de Trabalho (*Work List*)

É uma lista de trabalho associada com um determinado participante de *Workflow* (ou em alguns casos com um grupo de participantes de *Workflow* que podem compartilhar uma lista de trabalhos comum).

### 5.3.13 Aplicação Solicitada

É uma aplicação de *Workflow* que é invocada pelo Sistema Gerenciador de *Workflow* para automatizar, completamente ou parcialmente uma atividade ou ajudar um participante de *Workflow* no processamento de um item de trabalho.

O diagrama da Figura 5.1 mostra a relação geral entre alguns conceitos inerentes a *Workflow* descritos nos itens acima.

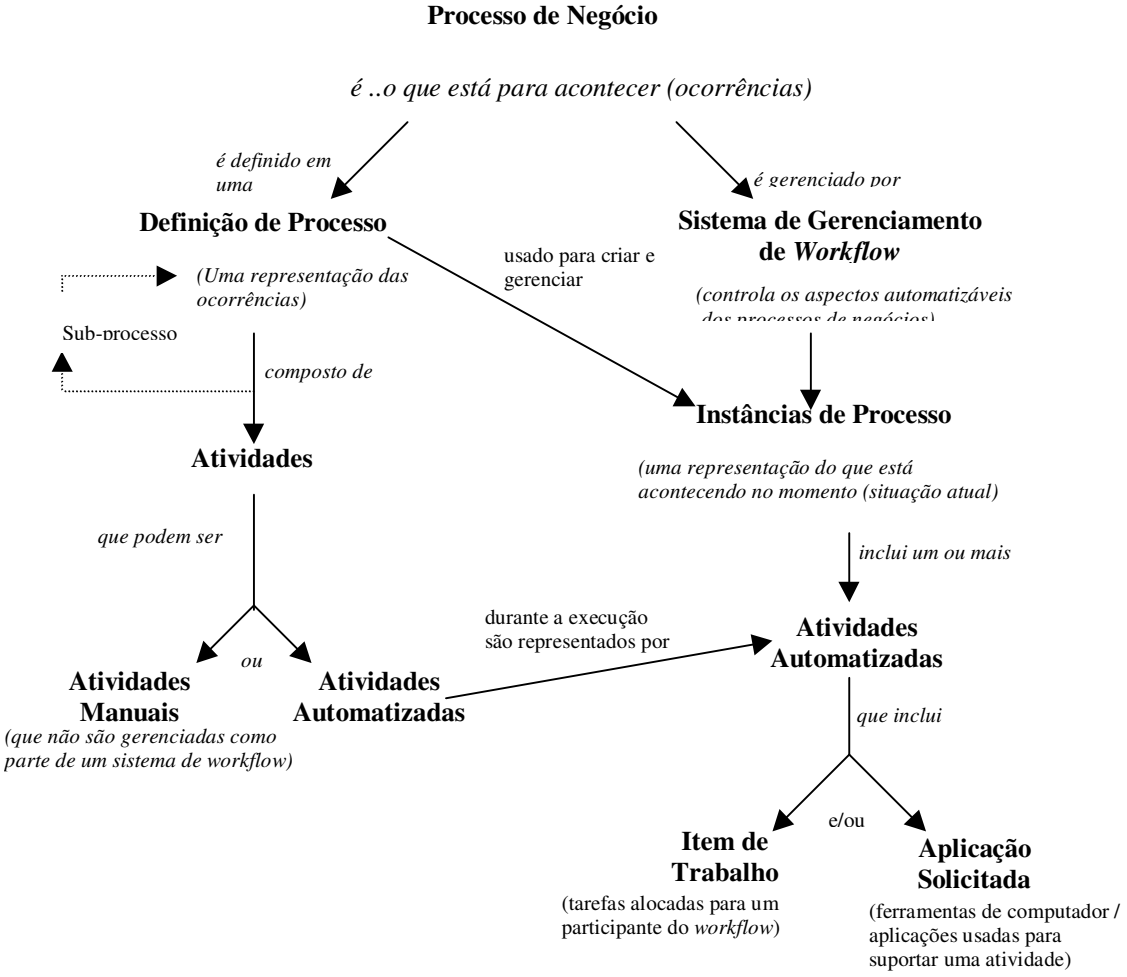



Figura 5.1 - Relacionamento entre a terminologia básica de *Workflow* (*WORKFLOW MANAGEMENT COALITION*, 1999)

### 5.3.14 Gatilho (*Trigger*)

Um evento ‘e’ dispara uma atividade ‘a’ se a ocorrência de ‘e’ causa a execução de ‘a’. Deve-se observar que ‘e’ pode ser um evento, uma atividade ou um ator, mas ‘a’ é sempre uma atividade.

O Diagrama Entidade-Relacionamento descrito em (BARTHELMES, 1997) representa a relação entre as noções de evento, objeto, atividade, processo e ator, e é útil para o entendimento da composição de um *Workflow* e os relacionamentos entre os elementos que o compõe.

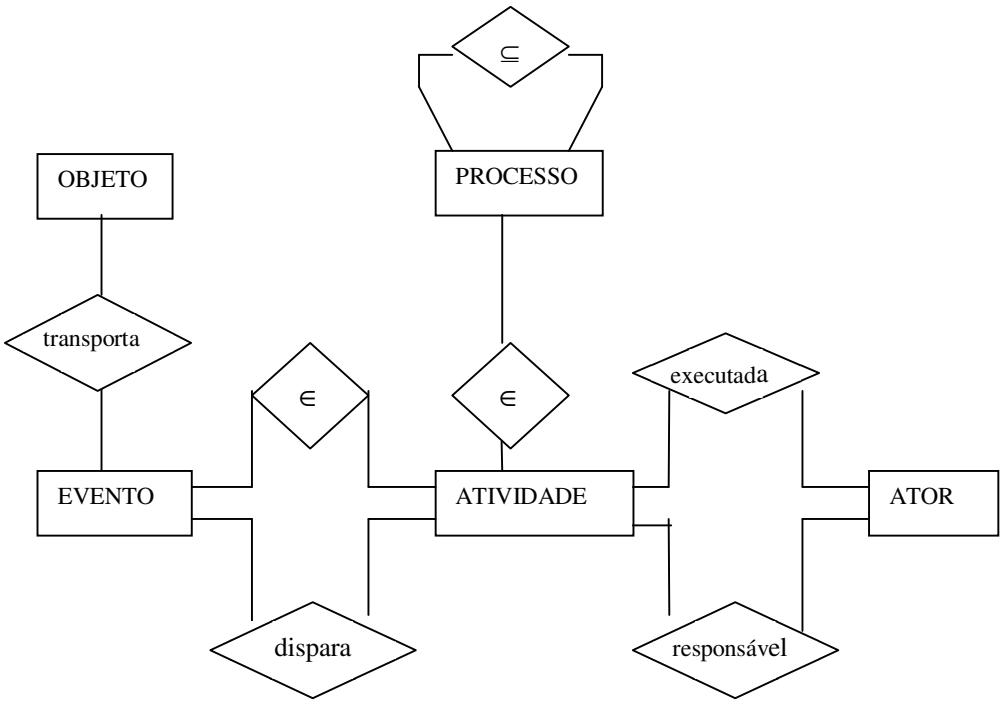



Figura 5.2 – Modelo Entidade-Relacionamento (BARTHELMES, 1997)

Ao analisar este diagrama, verifica-se que:

- Um ator é responsável por uma atividade;
- Uma atividade é executada por um ator;
- Um evento traz consigo (carrega) objetos;
- Uma atividade é disparada por um evento;
- Um evento pertence a uma atividade;
- Uma atividade pertence a um processo;

- Um processo é composto de um ou mais processos.

### 5.3.15 Tarefa

Uma parcela do trabalho para ser feito por um ou mais recursos num intervalo de tempo pré-determinado. Ex.: exame clínico por um médico, digitação de uma carta pela secretária.

### 5.3.16 *AND-Split*

Um ponto dentro do *Workflow* onde uma única linha de controle se divide em duas ou mais atividades paralelas, existindo assim um sincronismo para o início das mesmas.

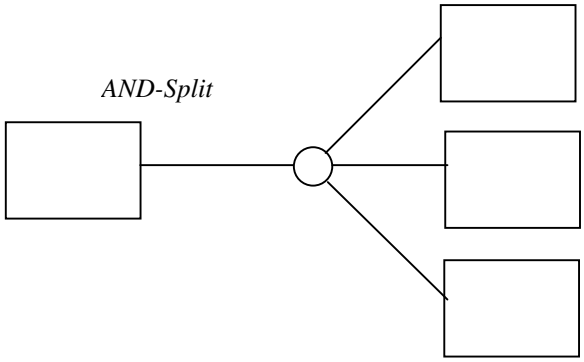



Figura 5.3 – Exemplo de AND-Split (*WORKFLOW* MANAGEMENT COALITION, 1999)

Na Figura 5.3 as três atividades ocorrerão paralelamente, apenas após a execução da atividade anterior.

### 5.3.17 *AND-Join*

Um ponto no *Workflow* onde duas ou mais atividades executando em paralelo convergem em uma única linha de controle comum, existindo um sincronismo para execução desta atividade.

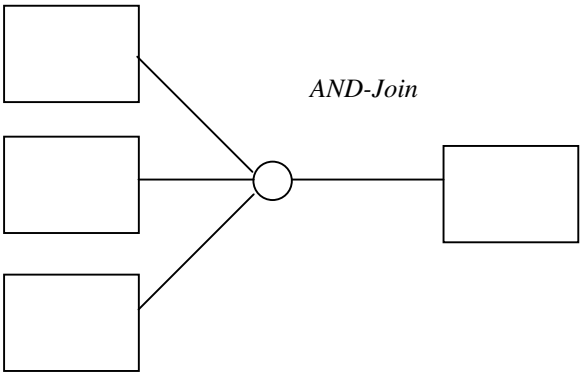



Figura 5.4 – Exemplo de *AND-Join* (*WORKFLOW MANAGEMENT COALITION*, 1999)

Na Figura 5.4, a atividade ocorrerá apenas após o término das três atividades antecessoras.

### 5.3.18 *OR-Split*

Um ponto dentro do *Workflow* onde uma única linha de controle faz uma decisão sobre qual desvio irá tomar com múltiplas alternativas de desvio do *Workflow*.

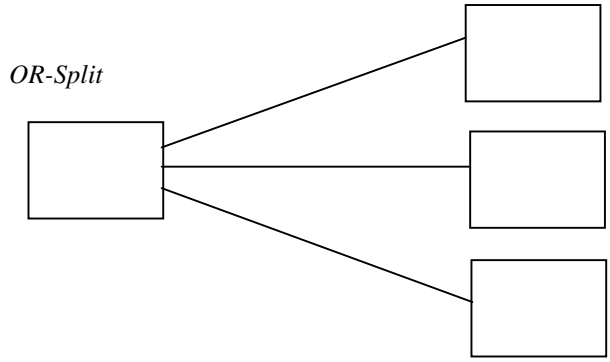



Figura 5.5 – Exemplo de *OR-Split* (*WORKFLOW MANAGEMENT COALITION*, 1999)

Na Figura 5.5, uma das três atividades ocorrerá após o término da atividade antecessora.

### 5.3.19 *OR-Join*

Um ponto dentro do *Workflow* onde duas ou mais atividades desviadas de um *Workflow* reconvergem em uma única atividade comum como o próximo passo dentro do *Workflow* (como nenhuma execução de atividade paralela aconteceu no ponto de junção, nenhuma sincronização é requerida).

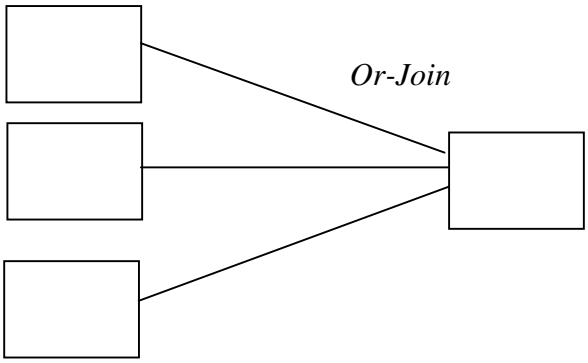



Figura 5.6 – Exemplo de *OR-Join* (*WORKFLOW MANAGEMENT COALITION*, 1999)

Na Figura 5.6, a atividade ocorrerá quando uma das atividades antecessoras ocorrer.

## 5.4 Competências de *Workflow*

*Workflow* cumpre com eficiência os seguintes propósitos (AMARAL, 1997):

- roteamento do trabalho: existe uma predefinição da seqüência das atividades que serão executadas. O participante recebe um item de trabalho e, quando o processamento deste é terminado, a tarefa seguinte pode ser iniciada. O roteamento pode ser baseado na resposta de uma determinada ação pelo usuário, ou pode ser baseado em regras, que depende da condição dos objetos manipulados após o processamento da atividade corrente;

- invocação automática de aplicativos: através do sistema de gerenciamento do *Workflow* um aplicativo pode ser invocado automaticamente se está associado a alguma atividade do atual participante do *Workflow*;

- distribuição dinâmica do trabalho: esta distribuição pode ser feita pelo sistema de gerenciamento do *Workflow*, através de um algoritmo ou manualmente por algum usuário com privilégios necessários para tal;

- priorização do trabalho: pode ser atribuído dinamicamente as prioridades às instâncias de *Workflow*, visto que algumas devem possuir uma prioridade superior às demais;

- acompanhamento do trabalho: capacidade de acompanhar determinada instância e descobrir imediatamente o seu *status* atual de processamento, sob a responsabilidade de quem está no momento, e quanto tempo ela está esperando a atividade atual.

## 5.5 Vantagens de *Workflow*

Ao disponibilizar todas as funcionalidades acima descritas, *Workflow* proporciona diversas vantagens e, dentre essas, podemos destacar (AMARAL, 1997):

- integração das atividades da empresa: as atividades desenvolvidas por divisões diferentes da empresa podem ser conectadas, compondo um único processo. Isso também pode ser feito através da *World Wide Web*, onde estas divisões podem estar em lugares físicos diferentes;

- garantia da integridade do processo: com o uso do Sistema Gerenciador de *Workflow* existe uma garantida de que as regras do processo não serão desrespeitadas, e que a ordenação das atividades será seguida;

- redução de tempo na execução do processo: como os processos são executados em uma ordem pré-estabelecida, sem perda de tempo com outras funções, o espaço de tempo em que um estado permanece pendente é virtualmente

eliminado. Assim, só são executadas tarefas necessárias à execução do processo;

- redução da redundância de atividades: como cada participante já tem organizadas as tarefas que dizem respeito a ele (são únicas), a repetição de tarefas não ocorre, tornando nula a redundância de atividades.

As vantagens proporcionam a redução do custo do produto final do *Workflow* e a qualidade do produto final do *Workflow*.

## 5.6 Arquitetura de um Sistema de *Workflow*

Apesar da grande variedade de produtos de *Workflow* encontrados no mercado, é viável conceber um modelo genérico de implementação de sistemas de *Workflow* que abranja a grande maioria desses produtos. Na arquitetura apresentada pela WfMC os principais componentes de um sistema de *Workflow* e suas interfaces são identificados.

A arquitetura da figura 5.7 apresenta três tipos de componentes: componentes de *software* (que provê suporte para as funcionalidades do sistema de gerência de *Workflow*), dados de controle do sistema (que são utilizados por um ou mais componentes de *software*) e aplicativos e suas bases de dados (que não são parte do sistema de gerência de *Workflow* em si, mas podem ser invocados por ele como parte do sistema global de *Workflow*).

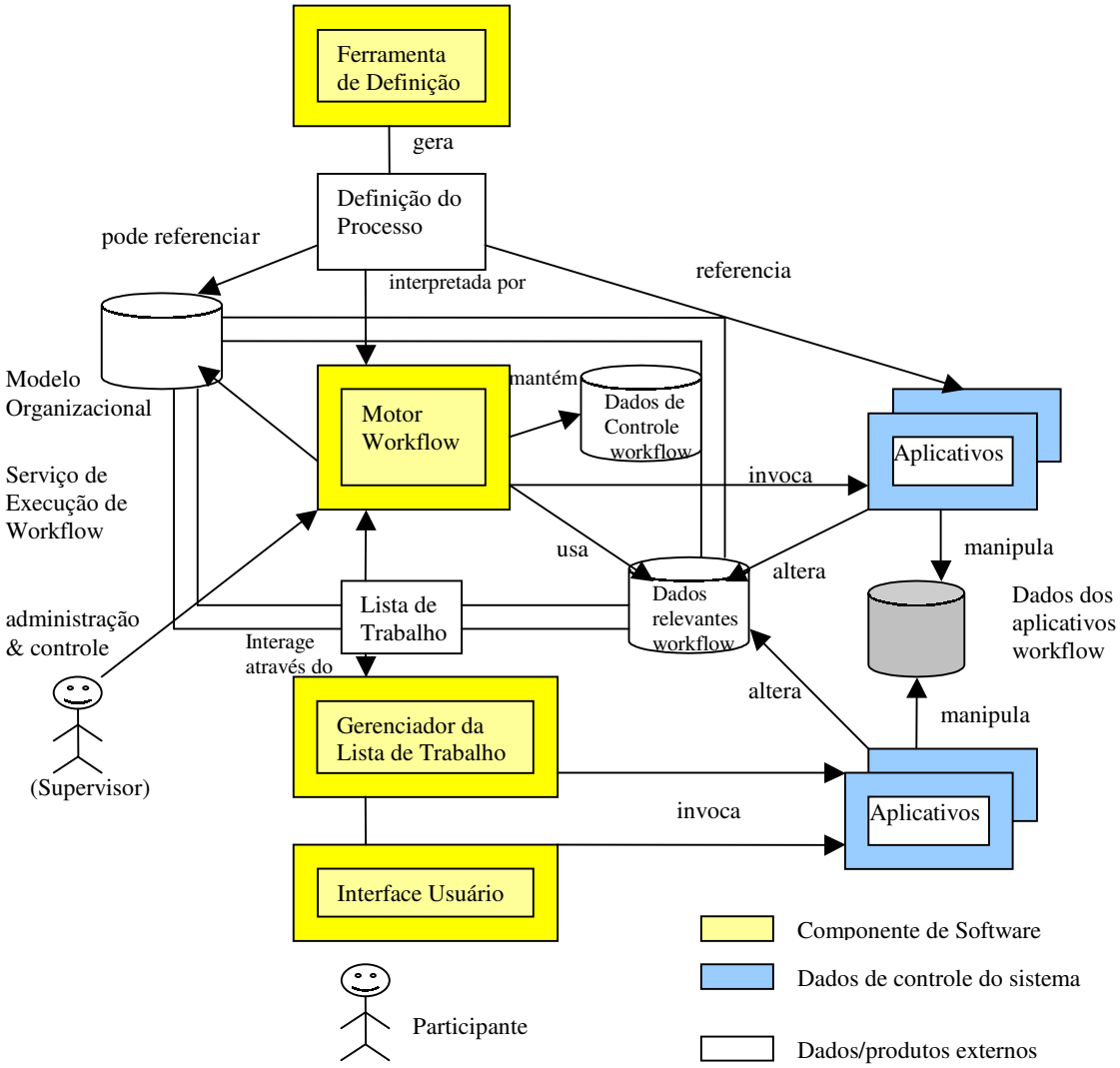



Figura 5.7 – Arquitetura genérica de um sistema de *Workflow* (*WORKFLOW MANAGEMENT COALITION*, 1999)

## 5.7 Meta-Modelo de *Workflow* Proposto pela WfMC

O Meta-Modelo proposto pela WfMC é considerado um modelo padrão que define processos. Através deste modelo, tem-se que o sistema de *Workflow* precisa ser provido com uma descrição do processo de negócio. Tal descrição compreende, basicamente, o nome, número da versão e condições de início e término do processo (*WORKFLOW MANAGEMENT COALITION*, 1999). A Figura 5.8 detalha este modelo.

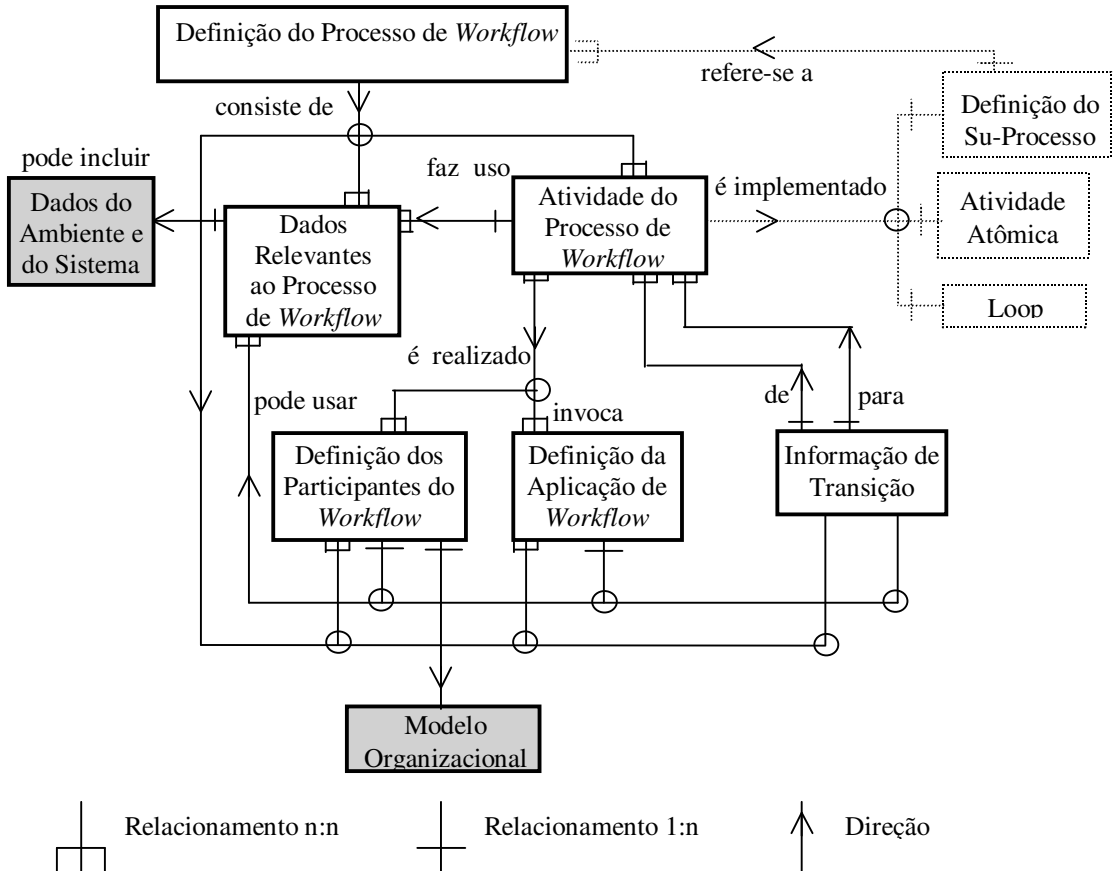



Figura 5.8 - Meta-Modelo de *Workflow* (*WORKFLOW MANAGEMENT COALITION*, 1999)

A definição de um processo consiste em atividades e dados relevantes à execução destas. Relacionados à definição do processo estão o participante, responsável pela execução de determinada atividade ou conjunto de atividades, e aplicações invocadas, necessárias para a execução de tais atividades. Além disso, na definição de um processo, pode haver referência para sub-processos. Por fim, o fluxo de controle dentro de um processo é determinado pelas informações de transição, normalmente, expressões *booleanas* baseadas em dados relevantes. (THOM, 2000)

Cabe ressaltar que, apesar das entidades, dados do modelo e/ou sistema e modelo organizacional, estarem incluídas no meta-modelo, estas são utilizadas, esporadicamente, pelo projetista. A entidade modelo organizacional, por exemplo, é

utilizada para obtenção de informações dos participantes do *Workflow*, tais como papel assumido na organização e nível de autoridade (THOM, 2000).

Na Tabela 5.1 são apresentadas as principais entidades do Meta-Modelo, com a descrição da função destas e principais atributos.

| Entidade | Função | Principais Atributos |
|---|---|---|
| Processo de *Workflow* | Identifica o processo de *Workflow* | Nome do processo<br>Descrição |
| Atividade de Processo de *Workflow* | Identifica as atividades que compõem o *Workflow* | Papéis que a executam<br>Aplicativos invocados<br>Pré-condições/pós-condições |
| Definição de Participante de *Workflow* | Define os participante e papéis que executam o processo | Custo<br>Estratégia de alocação de trabalho |
| Informações de Transição | Armazena os relacionamentos de dependência entre as atividades | Atividades predecessoras<br>Atividades sucessoras<br>Condições para a transição |
| Definição de Aplicação de *Workflow* | Descreve as ferramentas disponíveis para serem invocadas pelo WFMS | Aplicativo<br>Parâmetros utilizados |
| Dados Relevantes ao Processo | Dados gerados por uma determinada ação e que são necessários em próximas atividades, ou para uma transição ou para uma ferramenta invocada | Tipo de dado e valor |
| Modelo Organizacional | O WFMS pode consultar o modelo organizacional, a fim de obter informações sobre os participantes do *Workflow*, unidades organizacionais, níveis de autoridade e responsabilidade | Papéis e unidades organizacionais |

Tabela 5.1 – Funções das Entidades do Meta-Modelo de *Workflow*

## 5.8 Modelo de Referência para Sistema Gerenciador de *Workflow*

A WfMC propõe um modelo padrão a ser adotado por fabricantes de Sistemas Gerenciadores de *Workflow*, que exige que um *software* de *Workflow* possa se conectar a diversos Sistemas Gerenciadores de Banco de Dados, através de padrões de redes, com equipamentos de diversos fabricantes.

O modelo de referência apresenta uma abstração do serviço de execução de um *Workflow*, que pode representar qualquer tipo de ambiente de execução para um componente de controle de fluxo de aplicação de *Workflow*.

Neste modelo, são definidas cinco interfaces entre componentes, além de uma interface sobre o serviço de execução de *Workflow*, denominada WAPI (*Workflow API and Interchange Formats*). Esta interface consiste em uma série de construções pelas quais os serviços de execução de *Workflow* podem ser acessados. Desta forma, os serviços de *Workflow* podem ser implementados de diferentes formas, contanto que sejam oferecidas interfaces que traduzam os métodos internos de cada produto de *Workflow* para os métodos padronizados pela WfMC. (*WORKFLOW MANAGEMENT COALITION*, 1999)

A interação entre os Sistemas Gerenciadores e os componentes ocorre com o intercâmbio de definição de processo, o comportamento esperado por um processo em qualquer ferramenta de modelagem, a realização desse comportamento em qualquer infra-estrutura de execução de *Workflow* e interação entre aplicações e participantes.

Este modelo ainda engloba interfaces para interoperabilidade entre aplicações de *Workflow*, permitindo que aplicações desenvolvidas independentemente e gerenciadas por *Workflow* interajam na execução de processos complexos. (ALLEN, 2001). A Figura 5.9 apresenta o modelo de referência entre um Sistema Gerenciador de *Workflow* e o ambiente de seu contexto.

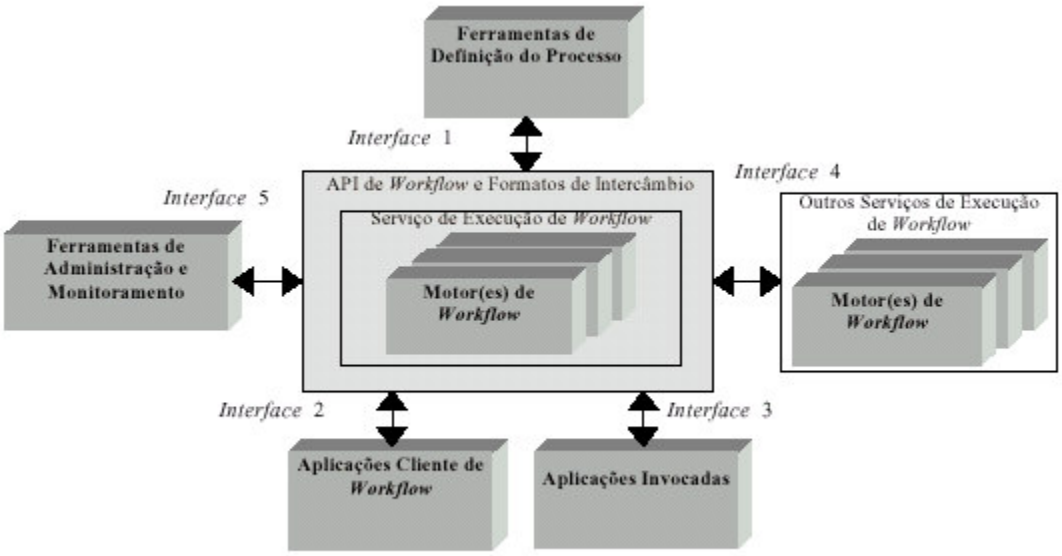



Figura 5.9 – Modelo de Referência de *Workflow* da WfMC

## 5.9 Modelagem de Sistemas de *Workflow*

Ao construir um sistema de *Workflow* está sendo modelada a maneira ou forma com que o trabalho é ou será executado (BARTHELMES, 1997, BARROS, 1997) e, portanto, deve ser levada em consideração a capacidade do sistema associar atores e papéis.

As técnicas de modelagem de *Workflow* devem ter, como objetivo básico, minimizar os problemas de coordenação do trabalho nos processos de negócios. Estas técnicas de modelagem baseiam-se no comportamento dinâmico do processo e precisam oferecer recursos para representação do fluxo de trabalho ao longo do processo modelado. (THOM & SCHEIDT, 1999)

Um dos maiores problemas da modelagem de sistemas de *Workflow* vem do fato que praticamente cada Sistema Gerenciador de *Workflow* (WFMS) utiliza sua própria técnica de modelagem, ou seja, não há um modelo conceitual amplamente aceito para a área de *Workflow*. Assim sendo, aparecem alguns problemas de interoperabilidade (por exemplo, WFMS diferentes querendo usar o mesmo modelo e este modelo podendo ser semanticamente igual e sintaticamente diferente) e problemas teóricos em relação ao

que é trabalho e como desenvolvê-lo dentro de um determinado ambiente. (NICOLAU, 1998)

Das diversas modelagens disponíveis, *Temporal Functionality In Objects With Roles Model* (TF-ORM) (EDELWEISS, 1998)(NICOLAU, 1998), Redes de Petri (BARROS, 1997), *Information Control Net* (ICN) (AMARAL, 1997), Modelo de Gatilhos (AKKERSDIJK, 1998), Casati/Ceri (BARROS, 1997), Barthelmes/Wainer (BARROS, 1997), *Action Workflow* (BARROS, 1997), Modelo de Interoperabilidade da WfMC (*WORKFLOW MANAGEMENT COALITION*, 1999), o presente trabalho usou a modelagem de Gatilhos com algumas inclusões (THOM & SCHEIDT, 1999) por atender aos requisitos do projeto (notação que permite representar primitivas de sincronismo, dependência entre as atividades, forma de disparo das atividades, pré-condições para a ocorrência de uma atividade), sua visualização ser de fácil entendimento em relação aos componentes de um *Workflow*.

O modelo de gatilhos representa atividades pertencentes a atores, onde o conjunto de atividades tem um propósito comum. As atividades relacionam-se entre si e, os gatilhos disparam atividades em função da ocorrência de eventos. (JOOSTEN, 1995)

Para a modelagem padrão do *Workflow* no ambiente SEA2 é usado o Modelo de Gatilhos com alterações, que segue a terminologia apresentada a seguir.

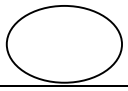
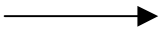
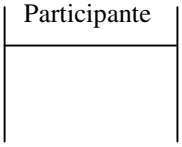
| Símbolo | Significado |
|---|---|
|  | Início e Fim de Processo |
|  | Atividade |
|  | Tomada de Decisão |
|  | Disparo de Atividade |
|  | Sincronização das Atividades<br>Na presença de triângulo preenchido, o *And-Split* e *And-Join* são representados, e quando não preenchido, o *Or-Split* e *Or-Join* são representados. |
| S(n) | S – seta<br>(n) – dígito indicando o número da seta |
|  | Cada coluna contém um participante e as sua lista de trabalhos. |

Tabela 5.2 – Terminologia do Modelo de Gatilhos com inclusões

Cada coluna de um processo modelado indica um participante do *Workflow* e a partir destas colunas pode-se verificar as *worklists* de cada participante.

A modelagem de um sistema de *Workflow* segue alguns passos: determinar o sistema de *Workflow*; determinar os participantes; identificar quais são as atividades executadas sob a responsabilidade de cada participante; verificar como cada atividade é disparada e criar o modelo. No ambiente SEA2 estas etapas são realizadas e, com a ajuda das KPAs de CMM, o sistema é modelado de acordo com suas prioridades.

## 5.10 Exemplo de Modelagem de *Workflow*

O exemplo de modelagem de *Workflow* que é apresentado neste item refere-se ao desenvolvimento de um *software* no Ambiente SEA2. *Workflow* existe independente do seu uso no desenvolvimento de aplicações, mas para o contexto desse trabalho, esse exemplo é o mais apropriado. A implementação do exemplo envolveu algumas fases (AKKERSDIJK, 1998):

- Descrição do sistema de *Workflow* a desenvolver;
- Identificação dos participantes do sistema de *Workflow* e respectivas atividades realizadas por cada um destes;
- Descrição das atividades;
- Modelagem do sistema de *Workflow*;

A seguir, essas fases são descritas.

### 5.10.1 Descrição do sistema de *Workflow* a desenvolver

O sistema deve criar e avaliar um componente do tipo tubo, que servirá como parte da estrutura do desenvolvimento de um *framework*. Mas, apenas para esta fase da modelagem, são criadas: a especificação estrutural da interface do componente, a especificação comportamental da interface do componente, a própria interface, o componente tubo e a avaliação desse componente.

### 5.10.2 Identificação dos Participantes e Atividades do Sistema de *Workflow*

Com o início da modelagem do protótipo, foram identificadas as atividades e os seus participantes mostrados pela Tabela 5.3.

| Atividade | Participante |
|---|---|
| Criar Especificação Estrutural | Ator1 |
| Criar Especificação Comportamental | Ator2 |
| Elaborar Interface | Ator1 |
| Implementar Tubo | Ator1 |
| Avaliar Tubo | Ator2 |

Tabela 5.3 – Atividades e seus participantes

### 5.10.3 Descrição das Atividades

Para uma melhor documentação das características das atividades e maior clareza dos processos durante o processo de implementação, cada atividade foi especificada com sua descrição (definição da atividade), participante (aquele que executa uma atividade), regra (restrição que determina o disparo de uma atividade), seta (o gatilho em si, indica a ação que faz com que uma atividade possa ser iniciada), ação (acontecimento que faz com que uma atividade possa ser iniciada) e forma de disparo (recurso utilizado na programação para iniciar uma atividade ou indicar que esta pode ser iniciada). Por exemplo, a atividade Criar Especificação Estrutural, que está descrita a seguir.

Atividade: **Criar Especificação Estrutural**

Descrição: O Ator1 deve estabelecer o canal de comunicação, listando os métodos requeridos e os fornecidos. O Ator1 deve verificar quais classes são necessárias para a estrutura da interface.

Participante: Ator1

S(1): Ação: Ativar sistema e permitir que o Ator1 inicie o processo.
Forma de Disparo: Escolher o *link* a ser associado à atividade.

### 5.10.4 Modelagem do Sistema de *Workflow*

A Figura 5.10 traz a modelagem do processo para elaboração de um componente do tipo Tubo.

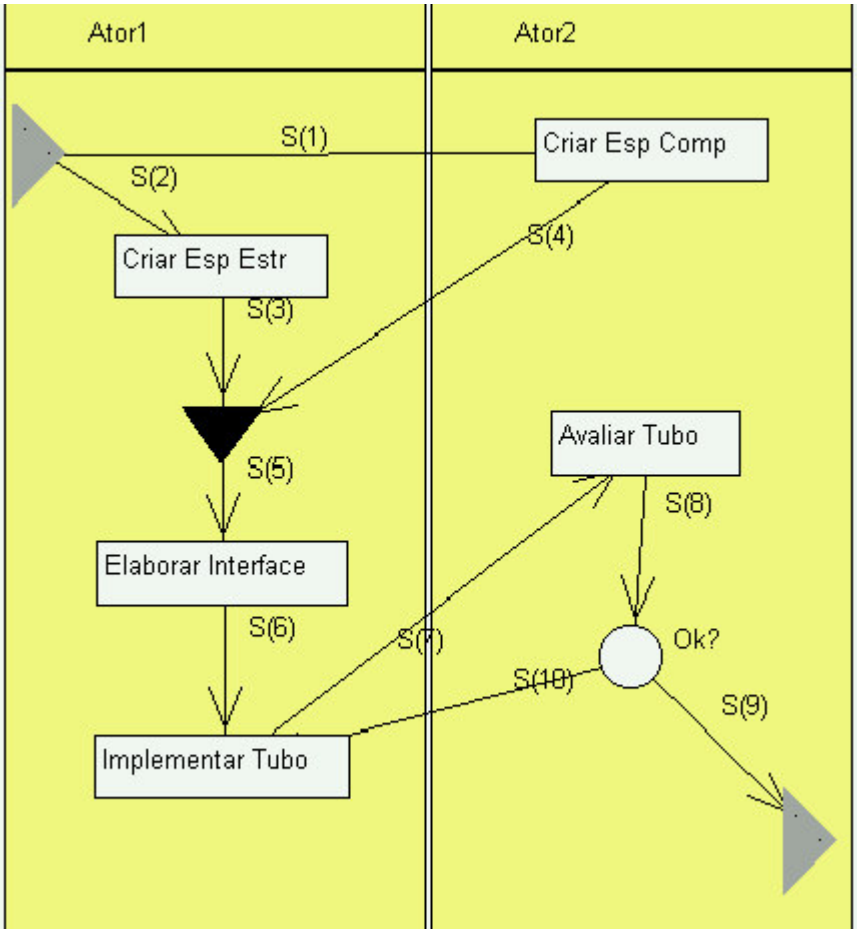



Figura 5.10 – Modelagem de desenvolvimento de um componente do tipo Tubo

Os atores "*Ator1*" e o "*Ator2*" iniciam o processo de desenvolvimento de componente do tipo Tubo. Ambos recebem o disparo inicial e processam a sua tarefa. As atividades "*Criar Especificação Estrutural*" e "*Criar Especificação Comportamental*" estão associadas (através de um *link*) a um documento que é desenvolvido pelo ator correspondente. Esse documento pode ser uma especificação ou apenas um modelo de uma especificação.

Com a edição e conclusão do desenvolvimento do documento, e conseqüentemente da atividade, ambas atividades disparam um conceito do tipo "*Trigger*" ao conceito "*Synchronize*", pois para o "*Ator1*" elaborar a interface do

componente do tipo tubo é necessário que as duas atividades anteriores estejam completas. O "*Synchronize*" assume a obrigação de que as atividades foram paralelamente concluídas.

Quando a atividade "*Elaborar Interface*" estiver concluída, é disparado um "*Trigger*" para a atividade "*Implementar Tubo*". O "*Ator1*" implementa esse documento e é disparado outro "*Trigger*" à atividade "*Avaliar Tubo*", que com a premissa do conceito "*Decision*" determina qual deve ser o andamento do processo: terminar ou reavaliar/refazer a atividade "*Implementar Tubo*". No caso de refazer a atividade, o fluxo do *Workflow* volta para a atividade "*Implementar Tubo*". Se a avaliação retornar Ok, é encerrado o *Workflow*.

A modelagem de um sistema de *Workflow* propõe um melhor entendimento do projeto, bem como um melhor controle das etapas que estão ocorrendo em função de um projeto.

# 6 *WORKFLOW* E CMM NO AMBIENTE DE DESENVOLVIMENTO DE *SOFTWARE* SEA

Este capítulo descreve a inclusão de *Workflow* e CMM no ambiente SEA. A tecnologia de *Workflow* e as premissas de CMM são introduzidas no ambiente através de um Editor de *Workflow* e de um Sistema Gerenciador de *Workflow* que, juntos, têm como principal objetivo modelar, organizar e gerenciar as etapas de desenvolvimento de um *software*.

A inclusão do Editor de *Workflow* e do Sistema Gerenciador de *Workflow* no ambiente SEA é possível pelo fato de o *framework* OCEAN (*framework* sob o qual o ambiente SEA foi desenvolvido) possuir uma estrutura extensível que permite a definição de diferentes estruturas de documentos, diferentes estruturas de visualização e variadas estruturas de edição de documentos.

A ferramenta de *Workflow,* introduzida no ambiente SEA, pode ser implantada em qualquer outro ambiente desenvolvido sob o *framework* OCEAN (diferente de SEA).

O novo ambiente desenvolvido é chamado SEA2 e possui as mesmas funcionalidades de SEA (descritas na Seção 3.2) com o acréscimo de gerenciamento, conformidade com KPAs de CMM no auxílio do gerenciamento, uso de ferramentas de *Workflow*, evolução automática dos processos de *Workflow* e gerenciamento em tempo real de execução de seus processos.

A seguir são descritos alguns trabalhos que tratam problemas semelhantes em relação ao gerenciamento de ambientes de desenvolvimento de *software* e é procedida uma comparação destes ambientes com o ambiente SEA2.

A Seção 6.2 apresenta algumas ferramentas comerciais de *Workflow* e, a partir da Seção 6.3 são detalhados os passos da inclusão de *Workflow* e CMM no ambiente SEA.

## 6.1 Trabalhos Correlatos

Alguns ambientes de desenvolvimento de *software* também sugeriram inclusão de ferramentas de gerenciamento à sua estrutura. Dentre esses, podemos destacar Transcoop, Prosoft, Estação Taba, Dynamit, Dukas e AgentFlow.

O projeto Transcoop (CORDENONZI, 2000) foi desenvolvido na UFRGS e atua na área de suporte ao desenvolvimento de *software*. Transcoop prevê a utilização de técnicas de gerência de projeto (baseados em *Workflow*), trabalho cooperativo e banco de dados para prover suporte ao processo de desenvolvimento de *software*. O modelo de processo de desenvolvimento de *software* representa os aspectos estáticos dos processos através de diagramas de pacotes e de classes. Os aspectos dinâmicos são representados por um conjunto de Diagramas de Transição de Estados que está associado a um elemento do modelo de estados, com restrições de integridade. O modelo prevê os possíveis estados, as transições de estados e a existência de interações humanas ao longo do desenvolvimento. Transcoop possui qualidade do produto (com subcaracterísticas da norma ISO/IEC 9126-1) e a qualidade de processo (com práticas-chave de CMM) agregados à sua estrutura. Comparando este ambiente ao novo ambiente SEA, SEA2, podemos dizer que o ambiente SEA2 apresenta os aspectos dinâmicos e estáticos no Diagrama de *Workflow*, descrevendo todas as atividades, seqüência dos processos e possíveis atores de cada atividade em uma única modelagem. Ambos ambientes usam atividades para representar os processos, (com estados particulares que definem o andamento dos mesmos) e que são executadas de acordo com uma determinada ordem. O gerenciamento do Transcoop (que está em fase de implementação)[10] deverá ser feito em tempo de execução, através do *Workflow*, consultando as informações do estado atual dos projetos mais o estado atual dos dados na metabase de dados. O gerenciamento do ambiente SEA2 ocorre em tempo real, com o acompanhamento da evolução dos estados dos processos.

[10] Informação recebida por e-mail em Outubro 2002.

O projeto Prosoft (LIMA, 1998) apóia a construção formal de ferramentas de *software* sob o paradigma algébrico. Seu gerenciador, que não impõe um paradigma de modelagem específico de processos, executa os processos descritos em um projeto. Os processos são um conjunto de atividades e as operações sobre um processo são refletidas diretamente nas suas atividades. Possui a vantagem de criação de uma biblioteca de definições de processos que podem ser reutilizados, economizando recursos e atingindo o nível 3 de maturidade de CMM. Comparando Prosoft ao ambiente SEA2, o gerenciamento no Prosoft é feito em tempo real, consultando os estados dos ATO's (ambiente de tratamento de objetos) e devolvendo esse estado à máquina de *Workflow*. As informações dos processos são consultadas e atualizadas durante a execução do mesmo (incluindo modificações dinâmicas no modelo). SEA2 também disponibiliza a evolução automática do *Workflow*, permitindo tempo real de processamento. Dentre as várias características do Prosoft, notou-se a falta de uma modelagem específica para suas especificações e objetos.

A Estação TABA (MACHADO, 2000) é um meta-ambiente, desenvolvido na UFRJ, capaz de gerar, através de instanciação, ambientes de desenvolvimento de *software* adequados às particularidades de processos de desenvolvimento e projetos específicos, com definição de processos, métodos e ferramentas CASE. Um ambiente na Estação TABA caracteriza-se pela descrição de uma seqüência de atividades, suas ferramentas de apoio, produtos de *software* gerados e recursos consumidos. O gerenciamento dos processos é feito através da identificação de atividades, gerência de execução das ferramentas e controle dos papéis e atividades dos usuários. TABA também apresenta o mapeamento entre a Norma ISSO/IEC 12207 e CMM tendo como base os objetivos das atividades. Comparando este ambiente ao ambiente SEA, ambos possuem semelhança no que diz respeito às atividades e às ferramentas acopladas aos ambientes. A linguagem definida nesses ambientes é voltada a objetos, porém SEA2 trata especificações (OO) e TABA trata componentes (que estão centrados no processo). Ainda em fase de implementação, TABA não executa seus processos em tempo real de execução, o que é efetuado no ambiente SEA.

O ambiente Dynamit (HEIMAN, 1997) gerencia processos no desenvolvimento de *software*. Este ambiente baseia-se em uma especificação formal, centralizado,

rodeado de tarefas, com gerenciamento de projeto. O ambiente Dynamit contém ferramentas para três regras: agenda para atribuir tarefas, gerentes do projeto com representação das tarefas e seus relacionamentos, e modelos de processos que definem um domínio de aplicação específico das partes do modelo e do ambiente. As tarefas são baseadas em um Diagrama de Transição de Estados, onde as transições são realizadas pelos gerentes e desenvolvedores ou pelos eventos disparados por regras. O gerenciador dos processos analisa as agendas e os trabalhos desenvolvidos para retornar algum estado a um processo. Em comparação ao ambiente SEA, o modelo Dynamit não possui modelos ou regras de qualidade associadas a ele e não leva premissas de *Workflow* na sua estrutura. Porém, o seu gerenciador trabalha em tempo real de execução.

O ambiente Dukas (VERAART, 1997) gerencia processos integrando regras SPICE e realidades do desenvolvimento de *software*. Dukas usa princípios de PSEE (Process Centered *Software* Engineering Environment) e *Workflow*. O ambiente implementa processos definidos (e outros criados) no modelo SPICE e introduz a eles novas práticas de trabalho. Os processos adaptados no modelo SPICE são divididos em práticas de trabalho, produtos de trabalho e relacionamento entre as práticas e os produtos. O modelo conceitual desse trabalho tem três componentes distintos: processo de *software* ideal, conjunto de atividades e conjunto de práticas e processos. O protótipo do Dukas usou a tecnologia www (Netscape Navigator, HTML e Delphi) para implementar os componentes. O gerenciador do Projeto do Dukas inicia uma atividade (com o início do script da atividade a ser processada) e, quando seu script estiver concluído, a próxima atividade inicia seu script (até o final do processo). As informações sobre as tarefas executadas retornam ao gerente de projetos, onde qualquer ação que restou é completada. As tarefas executadas são gravadas em um repositório, como instâncias (Practice Indicator Instance) e métricas são calculadas e armazenadas. Em comparação ao ambiente SEA, Dukas não usou regras de CMM e sim regras SPICE. O gerenciamento do Dukas não é feito automaticamente.

AgentFlow (CHEN, 2001) é um sistema gerenciador de processos dividido em duas partes: desenho dos componentes (PDE e FormDesign) e execução dos componentes (PASE e Agenda). Baseado no projeto PSEE e no projeto PASE (*Process Aided Software Engineering*) AgentFlow provê orientação, execução, automação,

diferentes visões do processo, ferramentas de invocação, retorno dos processos e uso de agendas. O modelo dos processos do AgentFlow suporta diferentes tabelas e diagramas para modelar processos de *software* que são guardados em um repositório. Uma atividade pode ter seu estado alterado dependendo dos estados dos artefatos (que são produtos gerados durante o desenvolvimento) e, a análise dos estados dos artefatos retorna as medidas/dimensões das atividades. Os dados referentes aos processos que foram inseridos no desenho dos componentes, as definições dos processos e os dados da aplicação são guardados em uma base de dados. Durante a execução dos componentes, a máquina de processos executa os processos de acordo com a definição da base de dados. As atividades são objetos desempacotados dentro de comandos SQL e guardadas dentro da base de dados. Em comparação com SEA, AgentFlow não possui CMM vinculado à sua estrutura, mas sim ISO9000. Ambos ambientes disponibilizam *Workflow* no desenvolvimento de processos. AgentFlow gerencia seus processos durante a execução dos componentes, através dos acessos às bases de dados.

A Tabela 6.1 esclarece as diferenças entre os ambientes de desenvolvimento de *software* pesquisados neste trabalho.

| Ambiente | Sistemas *Workflow* | Modelo de *Workflow* | Interação humana | O que trata o ambiente | Linguagem | Influência | Modelo no processo | Sistema Gerenciador | Processamento |
|---|---|---|---|---|---|---|---|---|---|
| **SEA2** | Sim | *Triggers* | Sim | Especificações OO, Interface de Componente e *Cookbook* | *SmallTalk* | OO e OOAD | CMM | Evolução automática do *Workflow* | Em tempo real |
| **AgentFlow** | Sim | PDE e Form Design | Sim | Dados guardados em Base de Dados e componentes | SQL | PSEE e PASE | ISO 9000 | Gerencia suas tarefas através de PASE e Agenda | Em tempo real |
| **TABA** | Sim | DTE, grafos e Redes de Petri | Sim | Componentes | C++, Eiffel e SGBD | Processo centralizado | ISO/IEC 12207 e CMM | Os dados que estão em um BD são lidos no início e salvos no término do processamento | Não é feito em tempo real de execução. |
| **Transcoop** | Sim | Estático: Diagr. Pacotes e DC Dinâmico: DTE e DCU | Sim | Meta base de dados | Modelagem de projeto em UML, não implementada. | Trabalho cooperativo e BD | ISO/IOEC 9126-1 e CMM | O estado do processo em desenvolvimento é identificado pela MBD e pelos DTE a cada momento | Está sendo implementado para ser em tempo real de execução |
| **Prosoft** | Sim | Sem modelagem específica | Sim | Objetos (ATO's – ambiente de tratamento de objetos) | Operações algébricas nos ATO's | OO + data driven + tipo abstrato de dados + método algébrico + modelos | CMM | Modificação dinâmica de processos | Em tempo real |
| **Dynamit** | Não | DTE e gráficos Pert | Sim | Documentos | Progress | Centralizado, especificação formal e rodeada de tarefas | Sem | Interação do gerenciador é feita com o acesso às agendas e tarefas | Em tempo real |
| **Dukas** | Sim | ----- | Sim | Componentes | www – Navigator, HTML e Delphi | PSEE e SPTME | Spice | A atividade processada retorna ao Manager seu estado que é gravado como uma instância no repositório | Não é em tempo real de execução. |

Tabela 6.1 – Comparação entre ambientes de desenvolvimento de *software*.

## 6.2 Ferramentas Comerciais de *Workflow*

Dentre algumas ferramentas comerciais de *Workflow*, podemos destacar (BRITTO, 2001):

FileNET's Pangaon *Workflow* Services: É uma ferramenta indicada, principalmente, para a automatização de um grande volume de processos. Além disso, apresenta várias funcionalidades para controlar, monitorar e avaliar a eficiência e produtividade dos processos. (FILENET, 2002)

COSA *Workflow*: Esta ferramenta possui um sistema de gerenciamento de *Workflow* que trata e distribui grande quantidade de tarefas. Possui uma ferramenta de edição para a construção de um modelo (que usa Redes de Petri para integrar com o nível de código), uma ferramenta para administrar o *Workflow* e uma ferramenta de simulação do *Workflow*. Um processo de negócio modelado de acordo com a rede de Petri contém atividades e estados. Quando a atividade é completada, o *token* é colocado no estado que ocorre após essa atividade. O *token* permanece na atividade até ela ser processada. O gerenciamento ocorre em tempo real de processamento, e as atividades podem ser processadas automaticamente caso a intervenção humana não seja necessária. (COSA, 2002)

BizFlow: Possui um analisador de processos (que gera relatórios para avaliação de performance do *Workflow*), um gerenciador de organograma (que gerencia os usuários na organização, atribuindo funções, estabelecendo regras e privilégios, determinando senhas, e-mails e arquivos de assinatura eletrônica), um gerenciador de aplicação (que permite definir as aplicações que serão utilizadas com o *Workflow*), um WebClient (que permite uma visualização do andamento dos processos, com tarefas listadas em cores diferentes, priorizando cada atividade pela sua cor e colocando-a nas primeiras posições da *worklist*), um Server (que gerencia o *Workflow* e suporta bases de dados e interfaces da WfMC), e uma ferramenta de modelagem. BizFlow possui evolução automática de seus estados, onde cada definição de processo é guardada em um arquivo XML. Toda vez que o processo é instanciado é feita uma cópia do arquivo XML. Toda instância de processo é sua própria cópia rodando. Qualquer problema em uma atividade só afeta a

sua instância, proporcionando escalabilidade ao sistema. O gerenciamento é feito em tempo real de processamento. (BIZFLOW, 2002)

TIB/InConcert: permite a modelagem dos processos de negócio; possibilita o gerenciamento e monitoramento, em tempo real, das atividades dos processos de negócio, possui arquitetura aberta, orientada a objetos, incluindo uma interface para programação com mais de 400 APIs e regras de processos de negócio; contém uma ferramenta para avaliação dos processos de negócio. (TIBCO, 2002)

FORO-WF: Tem como base arquitetura orientada a objetos. É compatível com CORBA, sendo que possibilita a integração com aplicações e bases de dados externas. Além disso, permite modelar, analisar, automatizar e controlar a execução dos processos de negócio através da máquina de *Workflow*. Os estados das atividades são alterados automaticamente, geralmente interagindo com sistemas externos e o seu gerenciamento é feito em tempo de processamento, sem interrupções. (FORO, 2002)

A Tabela 6.2 apresenta e compara as ferramentas comerciais de *Workflow* com o Sistema de *Workflow* implantado no ambiente SEA2.

| Ambiente | Modelo | Interação humana | Ferramenta gráfica para desenvolvimento | Especificação de data e hora para realização de uma atividade | Uso de Atividades e Worklist | Avaliação de eficiência e produtividade dos processos | Sistema Gerenciador | Processamento |
|---|---|---|---|---|---|---|---|---|
| **SEA2** | *Triggers* | Sim | Ferramenta própria | Em fase de implementação | Sim | Sim | Evolução automática do *Workflow* | Em tempo real |
| **FileNET's** | WfMC | Sim | MetaSoftware, IDS Prof. Sheer e Holosoft | Sim | Sim | Sim | Evolução automática do *Workflow* | Em tempo real |
| **COSA** | Redes de Petri | Sim | FlowModeling | Sim | Sim | Não disponível em (COSA, 2002) | Evolução automática do *Workflow* | Em tempo real |
| **BizFlow** | IDEF, Redes de Petri e outros | Sim | Process Designer Modeler | Sim | Sim | Sim | Evolução automática do *Workflow* | Em tempo real |
| **TIB** | Não disponível em (TIBCO, 2002) | Sim | Ferramenta própria | Sim | Sim | Sim | Evolução automática do *Workflow* | Em tempo real |
| **FORO** | WfMC e Modelo Transições | Sim | Ferramenta própria | Sim | Sim | Sim, em tempo de execução. | Evolução automática do *Workflow* | Em tempo real |

Tabela 6.2 – Comparação entre ferramentas comerciais de *Workflow*

## 6.3 Inserção de *Workflow* e CMM no Ambiente SEA

O ambiente SEA não possui a capacidade de criar e controlar o fluxo das atividades durante o desenvolvimento de um *software,* apenas suporta o aspecto tecnológico do desenvolvimento de *software,* permitindo a produção de especificações, verificação de consistência e geração de código. A técnica de *Workflow* e o modelo de CMM são inseridos nesse contexto para preencher a falta de uma modelagem de processos, a ausência de gerenciamento dos processos, e para prover qualidade no desenvolvimento de *software*.

O uso de *Workflow* ocorre com a inclusão da modelagem dos elementos de *Workflow* como extensão do *framework* OCEAN. Com essas extensões do *framework* (modelo, conceitos e figuras), é possível a criação do editor de *Workflow*, que usa outro *framework*, H*otDraw,* para a edição de suas figuras (BRANT, 1995).

Através da possibilidade de edição e manipulação dos elementos de *Workflow* (atividades, atores, disparos, sincronismos e decisões - descritos na Seção 5.3), alguns dos principais passos para o desenvolvimento de um sistema de *Workflow*, segundo (KROTH, 1997), são cumpridos: definição dos objetivos dos processos; definição do *Workflow* presente e definição da modelagem para o *Workflow* existente. Os detalhes da inclusão do Editor de *Workflow* no ambiente SEA estão descritos na Seção 6.5 e seus subitens.

O Sistema Gerenciador de *Workflow* do ambiente SEA2 segue os mesmos princípios de um sistema de Gerência de *Workflow* (Seção 5.3.3) e adiciona alguns requisitos de CMM (KPAs Planejamento do Projeto de *Software*, Acompanhamento e Supervisão do Projeto de *Software* e Garantia de Qualidade de *Software*) à sua estrutura.

Para o cumprimento dos procedimentos associados a KPA Planejamento do Projeto de *Software*, foi criado um Editor de *Workflow* que, entre outras aptidões, permite construir o plano de desenvolvimento do *software*, atribuir tarefas aos atores envolvidos no processo e ordenar a execução dos processos em um projeto. Na Seção 6.7.2 é descrita a associação feita entre as metas de CMM da KPA Planejamento do

Projeto de *Software* e os procedimentos incluídos no ambiente SEA2 para suprir essas metas.

Para atingir as metas da KPA Acompanhamento e Supervisão do Projeto *Software*, foi criado um Sistema Gerenciador *Workflow* que permite acompanhar e supervisionar a execução dos processos no desenvolvimento de um *software*. As atividades elaboradas no Diagrama de *Workflow* e disponibilizadas pelo Sistema Gerenciador de *Workflow* são visíveis e estendidas a todos participantes do projeto. Os participantes podem constatar o andamento das atividades, a ordem exata definida pelas dependências entre as atividades, a conclusão ou não de atividades, a necessidade de mudanças nas estimativas das datas de conclusão das atividades (caso seja observado o erro em alguma estimativa) e a necessidade ou não de incluir ou excluir atividades do projeto.

Para o cumprimento dos procedimentos associados a KPA Garantia de Qualidade de *Software*, que visa gerenciar a qualidade do processo de desenvolvimento de *software*, o Sistema Gerenciador *Workflow* permite a observação por parte de um gerente, das atividades, do modelo de *Workflow* proposto, do andamento de cada atividade e, principalmente, da conformidade entre o gerenciamento do processo e as políticas escritas para este projeto. Assim, pode ser feita revisão e auditoria das atividades.

A ferramenta de *Workflow* cobre diretamente as KPAs Planejamento do Projeto de *Software* e Acompanhamento e Supervisão do Projeto de *Software*, e parcialmente a KPA Garantia de Qualidade de *Software*, porém, outros requisitos de CMM podem ser associados às atividades de *Workflow*. Neste caso, um requisito pode ser a própria atividade do projeto. Por exemplo, a atividade "*Peer Review*", pertencente ao nível 3 de CMM, que executa os passos referentes a essa KPA.

O Sistema Gerenciador também possibilita o controle das decisões a serem tomadas e das decisões que foram tomadas. Assim, é possível saber qual atividade foi escolhida de acordo com uma premissa estabelecida por uma decisão.

O novo ambiente SEA2 introduz ferramentas de suporte a *Workflow*, gerenciamento de processos, planejamento e controle dos projetos. As ferramentas de gerenciamento contam com o auxílio do modelo de CMM e, portanto, pode-se dizer que o novo ambiente suporta parte do nível 2 de CMM, isto é, insere a capacidade de gerenciamento.

A seguir são descritos o Editor de *Workflow* e o Sistema Gerenciador de *Workflow*/SEA no ambiente SEA2.

## 6.4 A Estrutura de Especificação de *Workflow* no Ambiente SEA2

O primeiro passo para implementar *Workflow* no ambiente SEA, é a criação da estrutura de especificação de *Workflow* no *framework* OCEAN.

Uma especificação agrega elementos de especificação (instâncias de subclasses de *Concept* e *ConceptualModel*) e registra associações de sustentação e referência entre pares de elementos de especificação.

Para produzir uma estrutura de especificação deve-se definir um conjunto de tipos de modelo (subclasses concretas de *ConceptualModel*), um conjunto de tipos de conceito (subclasses concretas de *Concept*) e uma rede de associações entre os elementos desses conjuntos (SILVA, 2000b).

Para *Workflow*, foi criado um modelo, com base na modelagem de Gatilhos Aperfeiçoada (THOM & SCHEIDT, 1999), chamado *WorkflowDiagram* (subclasse concreta de *ConceptualModel*) e um conjunto de conceitos, *Actor, Activity, Decision, Trigger, Begin/End, Dart* e *Synchronize* (subclasses concretas de *Concept*) que são inseridos no *framework* OCEAN. A Figura 6.1 mostra a criação deste modelo e destes conceitos no *framework* OCEAN.

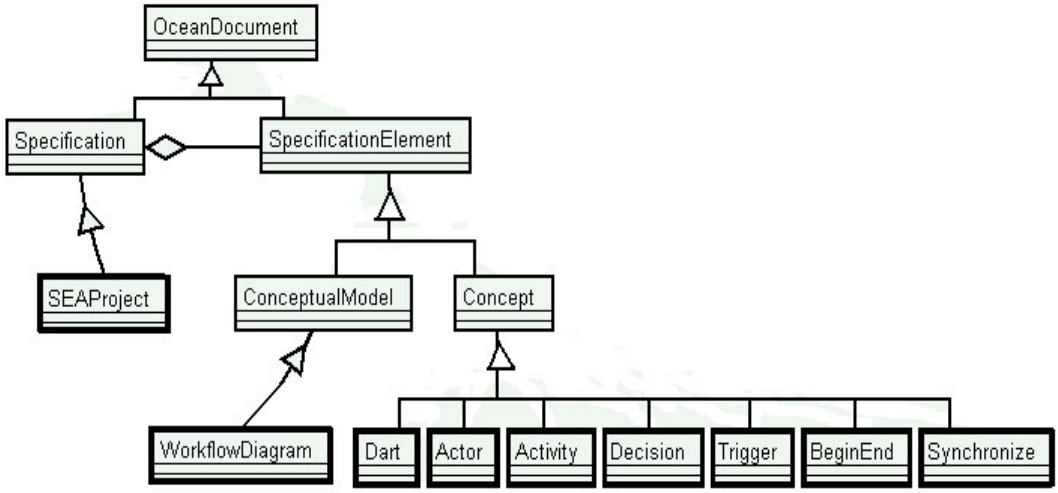



Figura 6.1 – Subclasses dos conceitos de *Workflow*

A partir das especificações elaboradas (modelos, conceitos e suas interligações) é criada a tabela de sustentação e referência das ligações semânticas entre os elementos de especificação (SILVA, 2000b). A tabela de sustentação e referência para os conceitos de *Workflow* é a seguinte:

| Elemento sustentador | Elemento sustentado |
|---|---|
| *Actor* | *Activity, Beginend, Decision, Synchronize* |
| *Trigger* | *Dart* |
| *Beginend* | *Trigger* |
| *Activity* | *Trigger* |
| *Dart* | - |
| *Synchronize* | *Trigger* |
| *Decision* | *Trigger* |

Tabela 6.3 – Elementos de *Workflow* e seus sustentadores

A Tabela 6.3 mostra a quem o elemento de especificação está condicionado. O elemento sustentado depende do elemento sustentador para permanecer no modelo. Por

exemplo, no caso de um conceito do tipo "*Beginend*" ser excluído do modelo, os conceitos do tipo "*Trigger*" que estão ligados ao "*Beginend*" são excluídos também. No caso de ser excluído um "*Actor*", todos os conceitos atribuídos para esse ator são excluídos também. Isso demonstra a relação de dependência existente entre estes conceitos.

| Elemento referenciador | Elemento referenciado |
|---|---|
| *Actor* | - |
| *Trigger* | *Dart* |
| *Beginend* | - |
| *Activity* | - |
| *Dart* | - |
| *Synchronize* | - |
| *Decision* | - |

Tabela 6.4 – Elementos de *Workflow* e seus referenciadores

A Tabela 6.4 mostra os elementos referenciadores e seus referenciados. O elemento referenciador depende da referência a outro elemento, o elemento referenciado. Por exemplo, o conceito *Trigger* referencia um *Dart*, onde para cada *Trigger* existe um *Dart* associado.

O armazenamento dos conceitos é centralizado, permitindo que sejam referenciados por um ou mais modelos ou por outros conceitos. Assim, qualquer alteração sobre um conceito específico, produz efeito sobre todos os elementos de especificação que o referenciam.

## 6.5 Editor de *Workflow* no Ambiente SEA2

Com a introdução dos conceitos, modelos e interligações de *Workflow* na estrutura de especificação do ambiente SEA, estas especificações ficam disponíveis ao ambiente, mas falta um manipulador gráfico para essas especificações. O *framework* OCEAN usa

uma classe gerenciadora (*ReferenceManager*) que relaciona cada elemento de especificação a um mecanismo de visualização do elemento. A representação visual é feita com o *framework HotDraw* (BRANDT, 1995). Este *framework* possibilita criar e editar elementos visuais como linhas, texto, composição de figuras e outros. Assim, os conceitos de *Workflow* ficam associados aos elementos gráficos do *HotDraw* e, portanto, cada conceito possui uma figura associada.

A classe *ReferenceManager* referencia a subclasse *SpecificationElementInterface* que define a estrutura genérica dos mecanismos de visualização de elementos de especificação. *SpecificationElementInterface* contém as subclasses *ConceptInterface* (que corresponde à estrutura genérica dos mecanismos de visualização de conceitos) e *ConceptualModelInterface* (que corresponde à estrutura genérica dos mecanismos de visualização de modelos). A Figura 3.2, do capítulo 3, apresenta a estrutura dos mecanismos de visualização de conceitos e modelos no *framework* OCEAN.

Para o Editor de *Workflow* foi incluída uma subclasse concreta em *ConceptualModel*, a classe *WorkflowDiagram*, e subclasses concretas em *SpecificationCompositeFigure*: *WActorFigure, WActivityFigure, WDecisionFigure, WSynchronizeFigure,, WBeginendFigure e WDartFigure*. Estas subclasses estão relacionadas diretamente às subclasses de *SpecificationElement*, pois existe a necessidade de uma associação entre a parte gráfica e a parte semântica do modelo, onde qualquer alteração feita graficamente altere ou não o respectivo contexto semântico. A Figura 6.2 exibe as subclasses inseridas no *framework* OCEAN para a criação do Editor de *Workflow*.

.

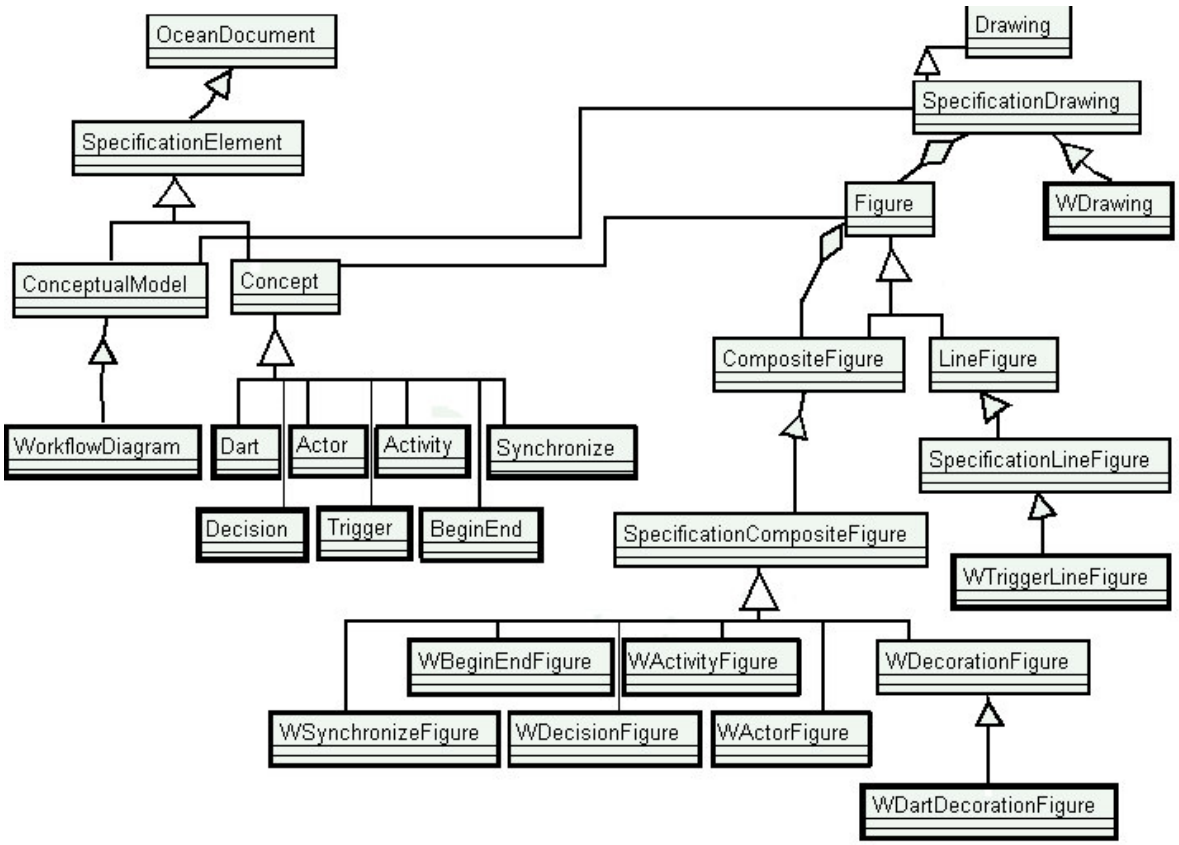



Figura 6.2 – Subclasses do Editor de *Workflow*

A partir da estrutura de interface criada e das figuras desenhadas, é possível ter uma visão do modelo de *Workflow* inserido no ambiente. A figura 6.3 mostra o Editor do Ambiente.

Os conceitos representáveis no Editor de *Workflow* do ambiente SEA são *Activity*, *Actor*, *Dart*, *Decision*, *Trigger*, *Beginend* e *Synchronize,* que foram apresentados na Tabela 5.2. O sustentador (SILVA, 2000b) dos conceitos, ou seja, o conceito imprescindível para um Diagrama de *Workflow* (na sua ausência nenhum outro conceito é criado) é *Actor*, o que o obriga a ser o primeiro a ser incluído em um Diagrama.

Normalmente, uma atividade dispara outra atividade que pode ser executada por outro participante ou não. Uma atividade pode ser acionada em paralelo com outra atividade. As setas indicam a ordem das atividades que, dependendo da decisão a ser tomada, alteram a ordem dos processos. Uma decisão tem um condicionador de ocorrência do tipo "if". O número de conceitos em Diagrama de *Workflow* varia dependendo de cada caso. Porém, o único que tem seu número delimitado é *Beginend* (2), pois um *Workflow* só tem um início e um fim.

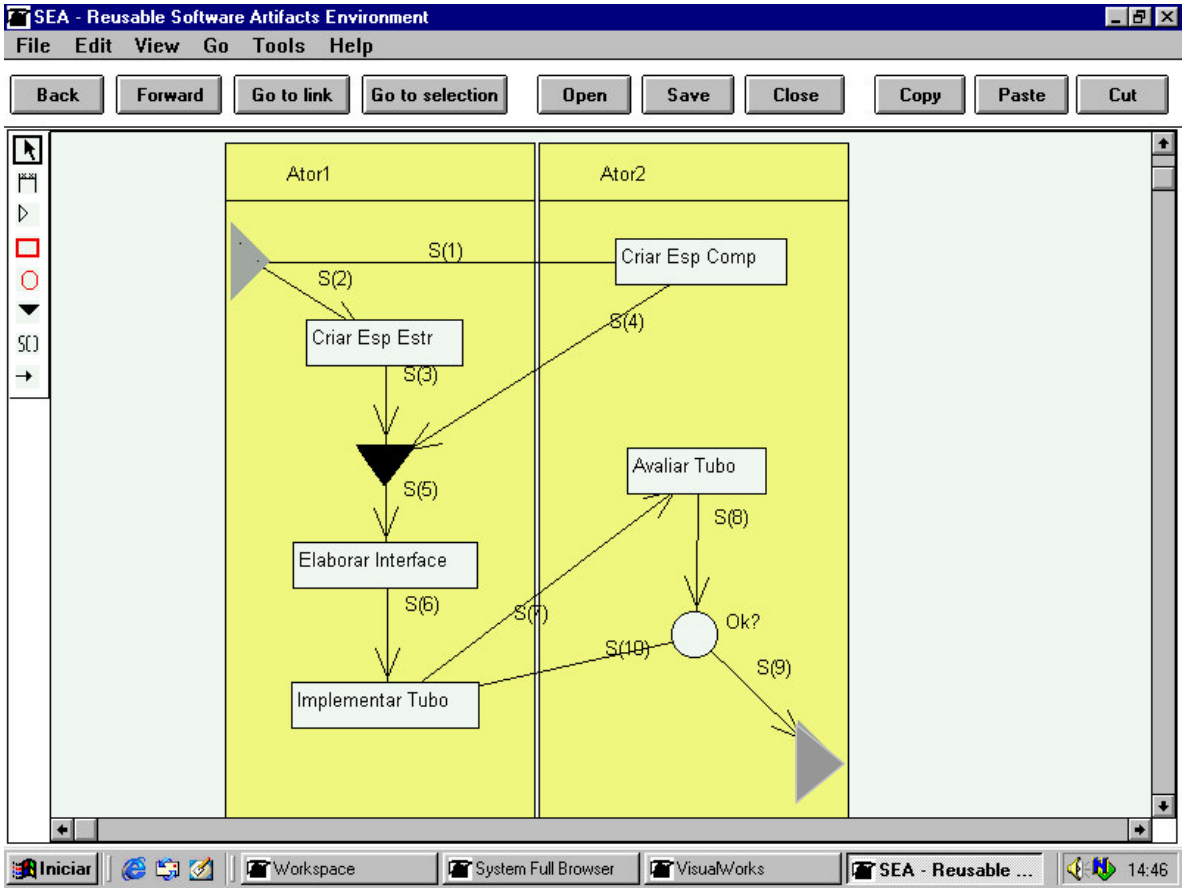



Figura 6.3 – Editor de *Workflow*

### 6.5.1 Janelas de edição dos conceitos de *Workflow*

Os conceitos de *Workflow* possuem janelas de edição próprias, pois existem diversas propriedades que precisam ser alteradas para garantir a consistência de cada conceito. A estrutura das subclasses inseridas no *framework* OCEAN para possibilitar a edição dos elementos pertencentes a um *Workflow* é apresentada na Figura 6.4.

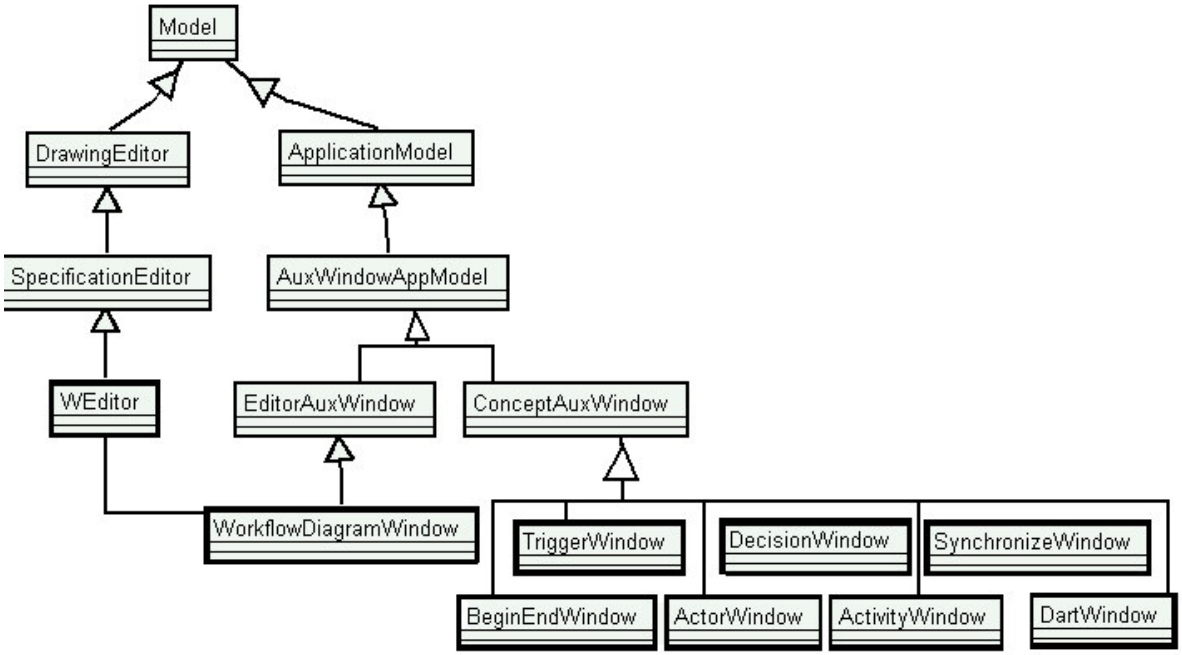



Figura 6.4 – Subclasses das janelas do Editor de *Workflow*

As janelas dos conceitos de *Workflow* são subclasses de *ConceptAuxWindow*, que por sua vez é subclasse de *ApplicationModel*.

Existe uma associação entre a classe *WEditor* (subclasse de *SpecificationEditor*, que é subclasse *DrawingEditor*) e a classe *WorkflowDiagramWindow* e ambas *DrawingEditor* e *ApplicationModel* são subclasses de Model.

A Tabela 6.5 mostra o que é permitido realizar com cada elemento.

| TIPO DE CONCEITO | O QUE A JANELA DE EDIÇÃO PERMITE |
|---|---|
| *Actor* | - alterar o nome do ator;<br>- escrever um texto explicativo sobre as tarefas do ator. |
| *Beginend* | -------- |
| *Activity* | - alterar o nome da atividade;<br>- escrever um texto explicativo sobre a atividade;<br>- criar o *link* para o documento a ser apontado.<br>- colocar datas prováveis de início e término da atividade.<br>- atribuir um esforço previsto para a atividade<br>- atribuir à atividade sua forma de ação: humana ou computacional. |
| *Trigger* | - mostra a atividade de origem e a atividade de destino;<br>- mostra o ator de origem e o ator de destino. |
| *Dart* | - alterar o nome do *dart*;<br>- escrever um texto explicativo sobre a ação da seta e sua forma de disparo. |
| *Decision* | - alterar o nome da decisão;<br>- escrever um texto explicativo sobre a decisão; |
| *Synchronize* | - mostra a(s) atividade(s) a ser(em) disparada(s) ao término do sincronismo |

Tabela 6.5 – Janelas de edição dos conceitos de *Workflow*

Uma atividade possui uma janela de edição, dividida em três páginas: *Identity*, *Attributes* e *Links*. A página *Identity* contém o tipo de conceito (*activity*), o nome da atividade (nome concedido ao ser planejado o *Workflow*) e o estado atual da atividade (que pode ser atividade processada, atividade não processada, atividade em andamento, atividade irregular). A página *Attributes* possui os atributos referentes a uma atividade. Entre esses atributos, alguns são obrigatórios: tipo de atividade (humana ou automatizada), descrição da atividade, data prevista para o início da atividade e data prevista para término da atividade. O esforço a ser medido (cálculo sobre o tempo e esforço que foram gastos para processar uma atividade) pertence a um trabalho futuro desse ambiente e, portanto, não é obrigatório para esta etapa do desenvolvimento de *software*. A página *Links* possibilita a edição do *link* a ser criado e escolha da especificação/documento ao qual a atividade está relacionada. O *link* é obrigatório para

todas as atividades consideradas automatizadas. É através do *link* que ocorre a ligação entre *Workflow*, o ambiente de desenvolvimento de *software* e o gerenciador de *Workflow*. As Figuras 6.5, 6.6 e 6.7 mostram as páginas de edição de uma atividade.

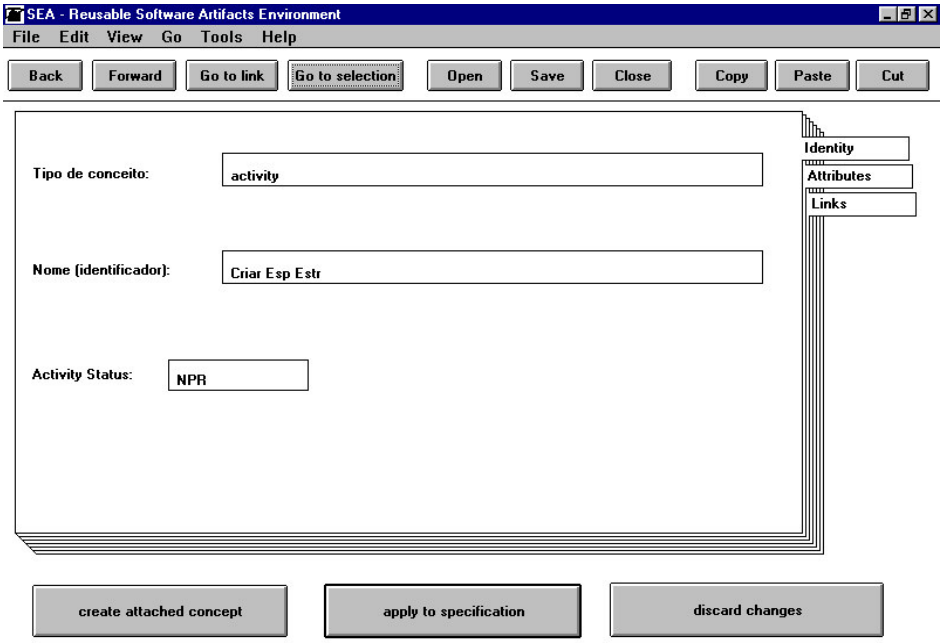



Figura 6.5 – Página *Identity*

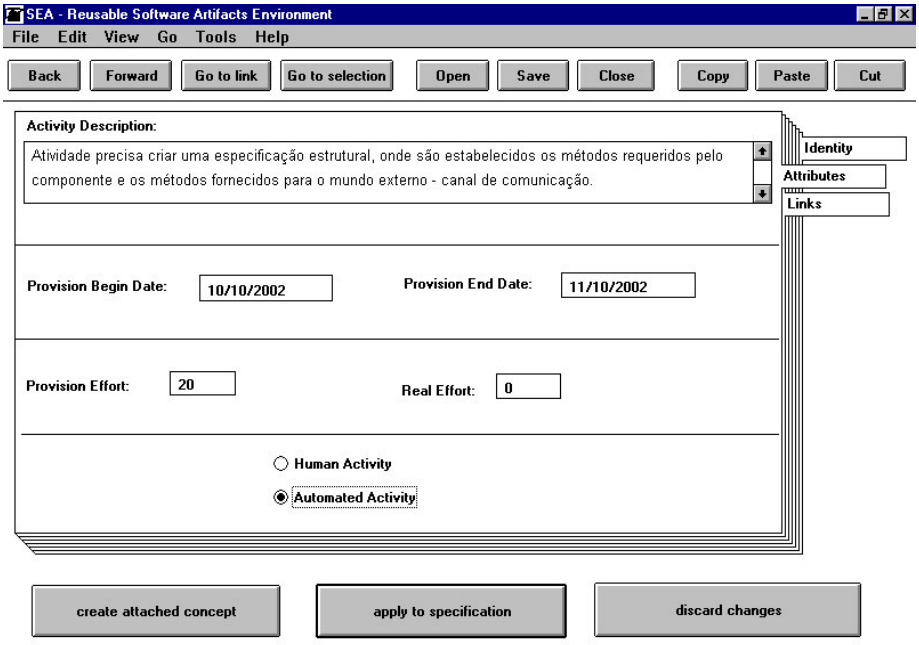



Figura 6.6 – Página *Attributes*

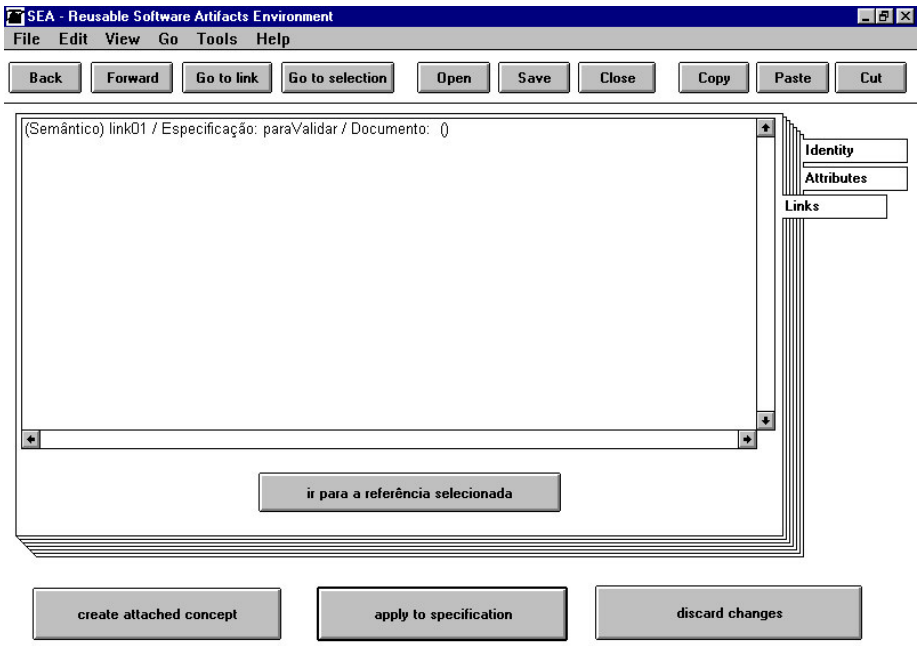



Figura 6.7 – Página *Links*

O ambiente SEA2 disponibiliza os seguintes recursos para testar a consistência de uma especificação (SILVA, 2000b):

- ações de edição bloqueadas: toda a ação de edição que envolva criação ou modificação de elemento de especificação é submetida à avaliação de conflito, que pode impedir a sua criação ou modificação. Alguns exemplos são a criação de atividades com mesmo nome, criação de atividades soltas (sem estarem ligadas a um ator), etc.

- ações de edição impossibilitadas: certas ações não são possíveis de realizar em função das interfaces de edição do ambiente SEA2. Por exemplo, não é possível criar "*Triggers*" sem ligá-los a um conceito. O editor não permite isso.

- ferramenta de análise: o ambiente apresenta uma ferramenta de análise para o modelo (*WorkflowDiagramAnalyserTool*) e uma ferramenta de análise para o conjunto da especificação (*ProjectAnalyser*). A ferramenta de análise do tipo de modelo avalia os tipos de conceito tratados no contexto do modelo

(estrutura e propriedades) e a avaliação dos inter-relacionamentos entre os conceitos referenciados. A ferramenta de análise de especificação invoca o analisador de modelo e verifica os aspectos que extrapolam os limites dos modelos.

As ferramentas de análise de *Workflow* produzem apenas um tipo de ação: descrição de erros encontrados (erros descritos no item 6.5.2), que são enviados a um arquivo de saída. A descrição de erros pode corresponder a imperfeições ou inconsistências do modelo, que devem ser alteradas para o modelo ficar consistente.

A Figura 6.8 mostra a janela de escolha dos analisadores. O analisador escolhido pode ser para o modelo (*Workflow Diagram Analyser tool*) ou para a especificação (*SEA Project Analyser*).

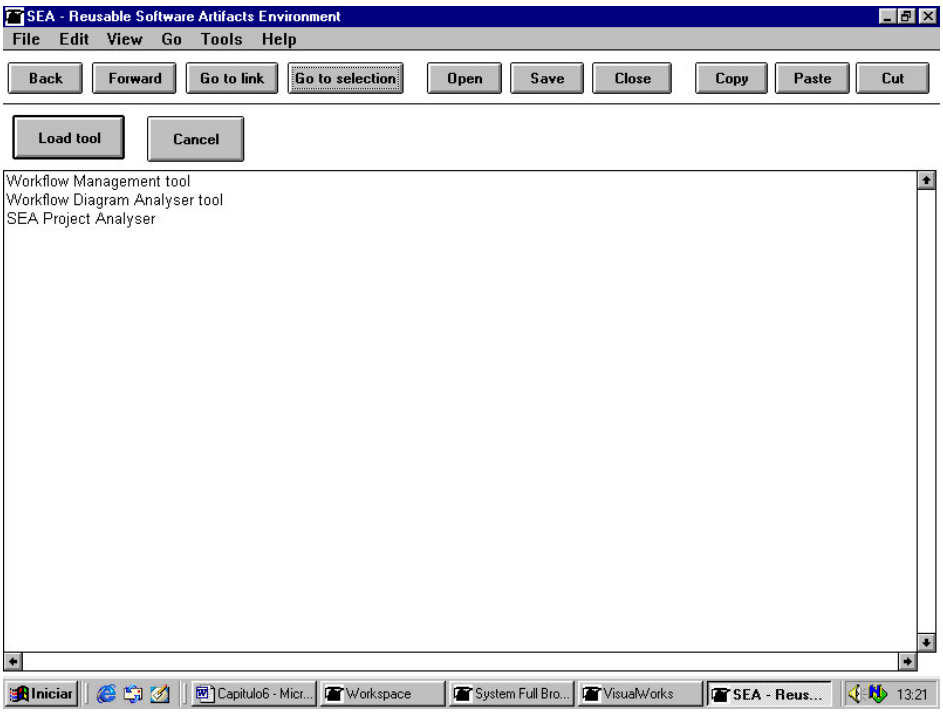



Figura 6.8 – Janela de escolha dos analisadores

Caso o analisador solicitado seja o analisador de modelos, o modelo descrito do Diagrama de *Workflow* corrente é analisado.

Tanto o analisador de modelos como o analisador de especificação levam à análise do modelo, pois uma especificação de *Workflow* somente possui um modelo associado.

### 6.5.2 O que é avaliado no Modelo de *Workflow*

O analisador de modelo de *Workflow* faz uma avaliação de modelo vazio e, depois, individualmente, avalia todos os conceitos inseridos no diagrama.

O primeiro teste, de modelo vazio, verifica a existência de atores no diagrama de *Workflow*. Qualquer elemento somente pode ser inserido no *Workflow* se existe um ator para este elemento.

O teste correspondente ao conceito "*actor*" verifica a existência ou não de atividades atribuídas a um ator. Caso o ator não possua nenhuma atividade, isso significa inconsistência no modelo, pois os atores precisam ter ao menos uma atividade para executar.

A análise feita com o conceito "*trigger*" testa a existência de conceitos soltos no modelo. Entre cada conceito deve existir, no mínimo, um *trigger* que interliga os conceitos e estabelece o caminho a ser seguido após o término de cada atividade. Através do "*trigger*" pode ser visualizado o andamento das atividades no projeto.

O conceito "*activity*" é o que merece mais atenção dentre todos os outros conceitos por ter seu conteúdo utilizado pelo gerenciador de *Workflow*. O analisador de atividades verifica a existência de um *trigger* disparando a atividade e um *trigger* sendo disparado pela atividade. Assim, tem-se controle de onde a atividade está chegando e para onde ela está indo. É verificada também a descrição de uma atividade, que contém os passos a serem seguidos para o processamento desta atividade. O teste referente ao *link* associado à atividade considerada automatizada também é obrigatório e muito importante, pois é através do *link* que se chega ao documento ao qual a atividade está relacionada. Ao saber qual o documento apontado pelo *link*, é possível saber o status

desse documento. Os testes específicos sobre um *link* são os seguintes: a atividade aponta para um único *link*, pois o documento que o *link* aponta reflete somente uma atividade, não existe mais de uma atividade apontando para esse mesmo *link*, pois as atividades são diferentes e conseqüentemente seus documentos também e, é testado se o *link* invocado já foi definido, pois não pode ser criado um *link* que aponte para um documento inexistente.

A análise feita com o conceito "*synchronize*" confere se este conceito está recebendo mais de um *trigger*, porque um sincronismo só ocorre se mais de uma atividade ou decisão o invoca. O sincronismo representa as várias ações que foram tomadas anteriormente para uma próxima ação ocorrer.

A verificação com o conceito "*decision*" testa a existência de uma descrição associada à decisão, que indica qual teste está sendo feito naquele momento. O resultado deve disparar no mínimo dois *trigger*s, que são as possibilidades de resposta para o teste. Esses dois *trigger*s também são analisados.

A análise de "*beginend*" verifica a existência de exatamente dois conceitos desse tipo no modelo. Isso porque existe um único "início do diagrama" e um único "fim do diagrama". Desses dois, o primeiro deve apenas disparar *trigger*s, e o segundo deve apenas receber um *trigger*. O analisador verifica essa consistência.

A análise do conceito "*dart*" confirma a existência de uma descrição associada a este conceito. Essa descrição informa qual a ação da seta e qual a forma de disparo da seta.

Caso ocorra algum erro de edição do diagrama de *Workflow*, o analisador de modelo indica, com o arquivo de saída de erros, o local exato de cada erro de cada conceito (através dos testes descritos acima). Assim, é possível corrigir e refazer o diagrama para torná-lo consistente.

### 6.5.3 O que é avaliado na Especificação de *Workflow*

O analisador de especificação de *Workflow* faz uma avaliação de modelo vazio (especificação sem modelo de *Workflow*) e, depois, realiza o teste de modelos pertencente à especificação.

No momento, existe somente um modelo de *Workflow* na especificação, por isso o analisador de especificação invoca o analisador do modelo de *Workflow*, que refaz todos os testes do item 6.5.2.

## 6.6 Restrições do Editor de *Workflow*

Existem algumas restrições referentes à edição dos elementos de *Workflow* no ambiente SEA2. Foi estabelecido que não é permitido que seja disparado um *trigger* de uma decisão para outra decisão. Uma condição desse tipo pode ser modelada de outra maneira. O mesmo ocorre com um disparo de *trigger* entre sincronismos. O editor não permite isso. As figuras abaixo mostram as outras possibilidades de edição para estas situações.

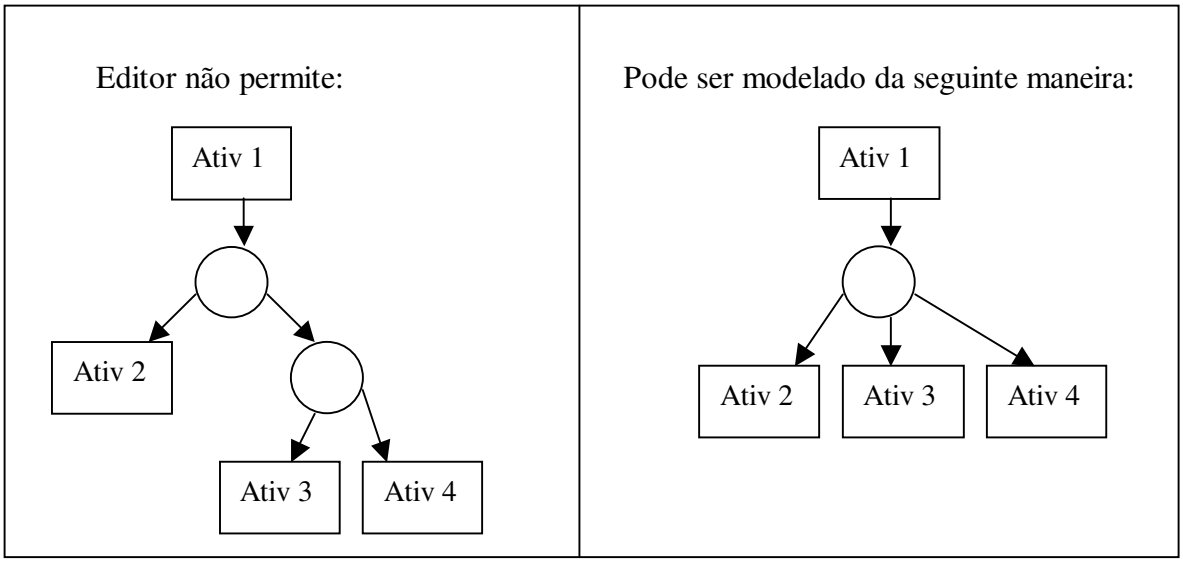



Figura 6.9 – Restrição de edição do elemento *Decision*

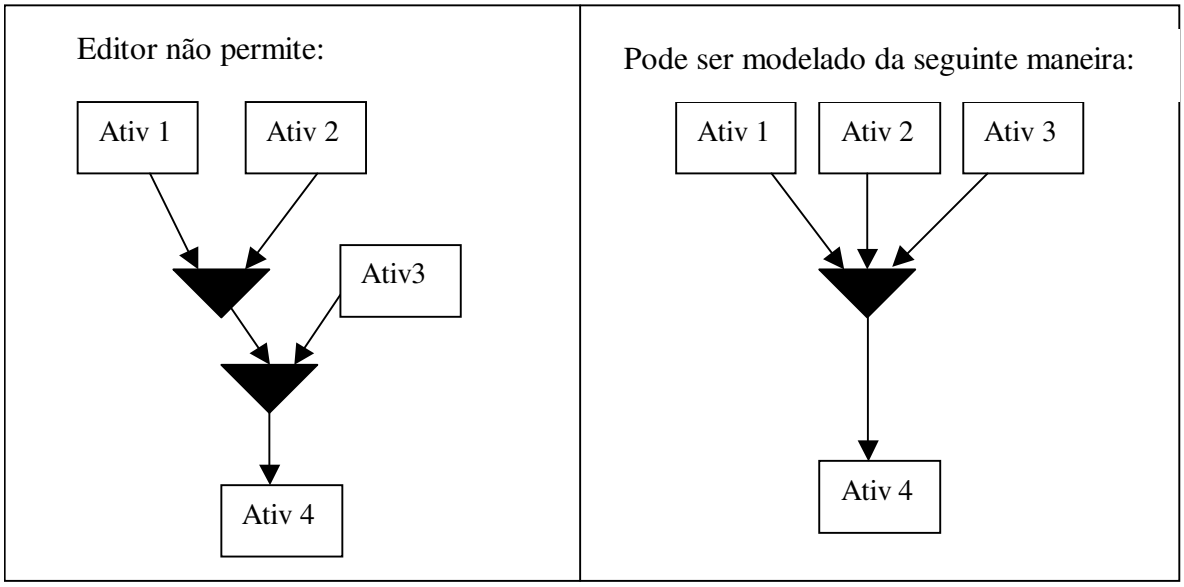



Figura 6.10 – Restrição de edição do elemento *Synchronize*

## 6.7 Sistema Gerenciador de *Workflow* no Ambiente SEA2

O sistema gerenciador de *Workflow* é inserido no ambiente SEA como uma ferramenta. As ferramentas são estruturas funcionais que reutilizam os procedimentos de edição semântica supridos pelo *framework* OCEAN.

As ferramentas são construídas como subclasses de *OceanTool* e são associadas a um ou mais tipos de especificação (para que foram construídas) e podem estar associadas a um ou mais tipos de modelos (se forem específicas para atuar sobre determinados tipos de modelo).

A ferramenta de gerenciamento do ambiente SEA2 chama-se *Workflow Management Tool*, que verifica e gerencia as atividades e processos realizados. O ambiente apenas permite que esta opção seja acionada se a especificação de *Workflow* estiver "*Frozen*" (estado da especificação – ou ela está *frozen* ou *not frozen*), e esta opção só é permitida se a especificação está validada. Assim, apenas especificações de *Workflow* consistentes podem ser gerenciadas. Na Figura 6.9, a opção "*Workflow Management tool*" corresponde à análise a ser feita dos processos.

A Figura 6.11 mostra uma das telas do Sistema Gerenciador *Workflow*/SEA2, a tela *Modeling*, que mostra o modelo de *Workflow* que está sendo gerenciado e o andamento dos processos.

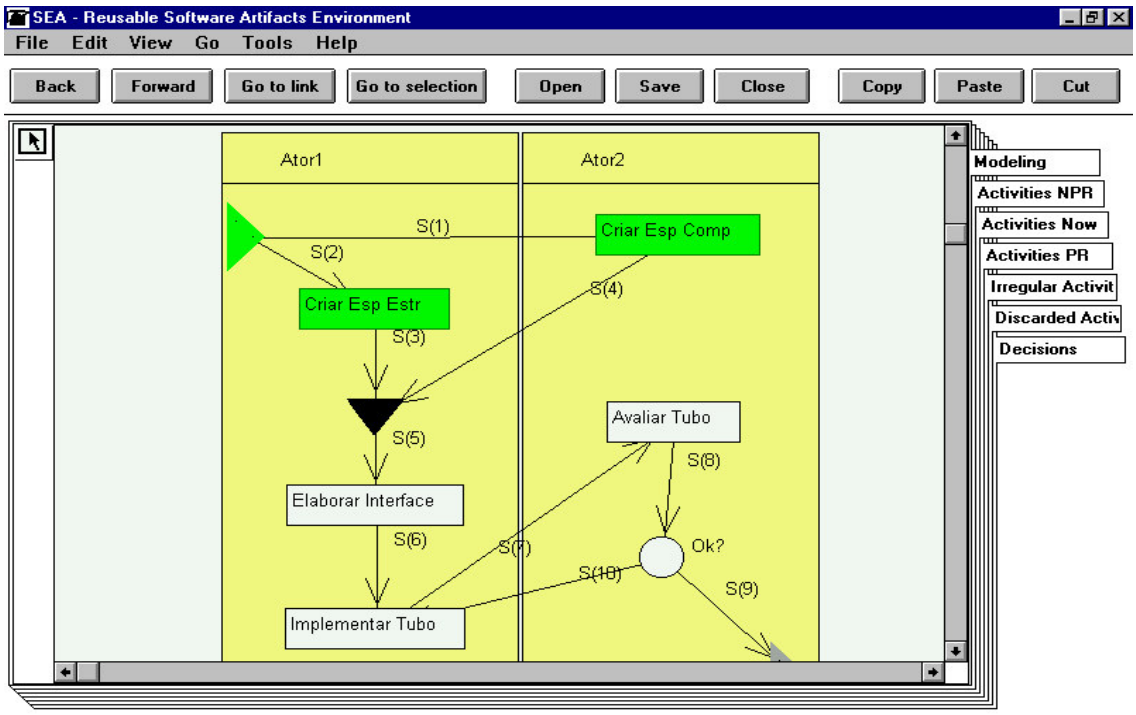



Figura 6.11 – Sistema Gerenciador de *Workflow* – Opção *Modeling*

O gerenciador de *Workflow* mostra os estados das atividades de acordo com cores: cor branca representa atividade não processada (NPR); cor azul representa atividade em andamento (EA); cor verde representa atividade processada (PR); cor vermelha representa atividade irregular (IR) e cor amarela representa atividade descartada (AD).

A figura 6.12 mostra a Máquina de Estados de uma atividade.

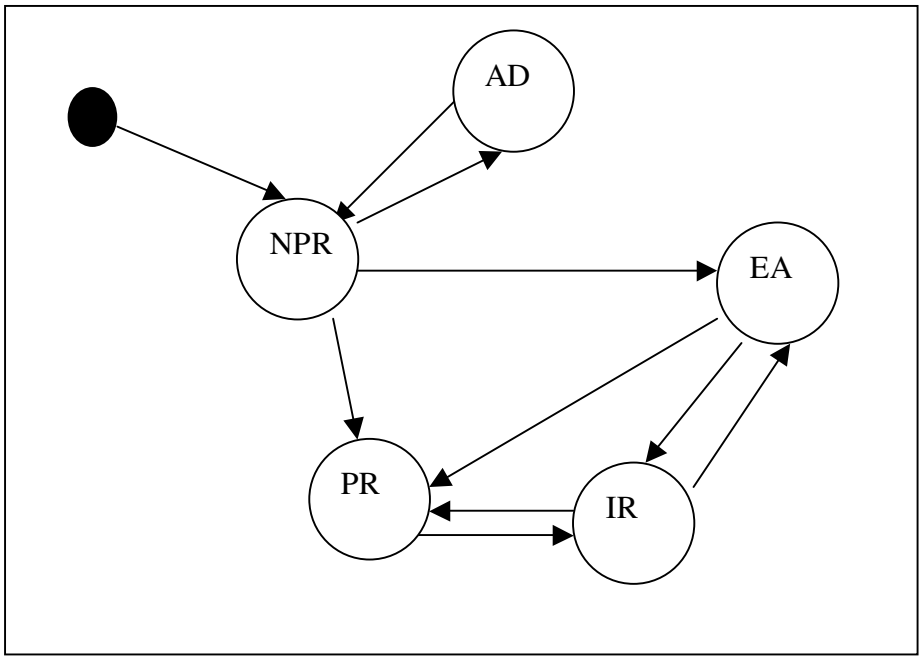



Figura 6.12 – Máquina de Estados de uma Atividade

Os estados de um documento interferem na classificação das atividades. Um documento pode estar "*not OK*", "*OK*" ou "*Frozen*". A Figura 6.13 mostra a Máquina de Estados de um documento.

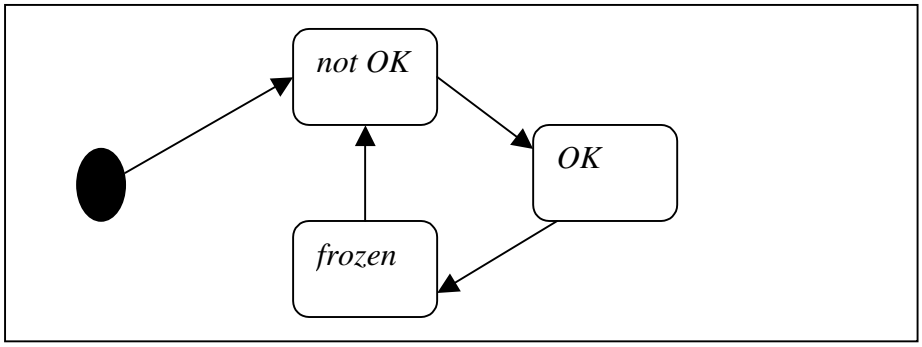



Figura 6.13 – Máquina de Estados de um Documento

Um documento considerado "*Frozen*" teve suas estruturas terminadas (prontas). Uma atividade que está associada (através de um *link*) a um documento "*Frozen*" é considerada "processada".

Caso a atividade aponte um documento considerado "*OK*" ou "*not OK*", mas que ainda não foi congelado (*Frozen*), esta atividade é considerada "em andamento". Isto

significa que o documento ainda não foi terminado e sua edição ou implementação está incompleta.

Se existe uma atividade com um *link* semântico, que não aponta nenhum documento, essa atividade é considerada "não processada", pois não existe parâmetro para saber como está esse documento.

Uma atividade pode ser considerada "irregular" no caso de possuir um *link* apontando um documento e este documento ser excluído do ambiente. Ou, no caso de existir um *link* na atividade considerada processada, e este *link* ser excluído e um novo *link* ser associado à atividade, gerando inconsistências.

No caso de ocorrer uma decisão entre quais atividades devem ser processadas e quais devem ser rejeitadas, as não escolhidas pela decisão recebem o estado de "descartada". Os estados das atividades são mostrados no Sistema Gerenciador através das cores na opção *Modeling*, ou através das janelas de opções dos estados das atividades.

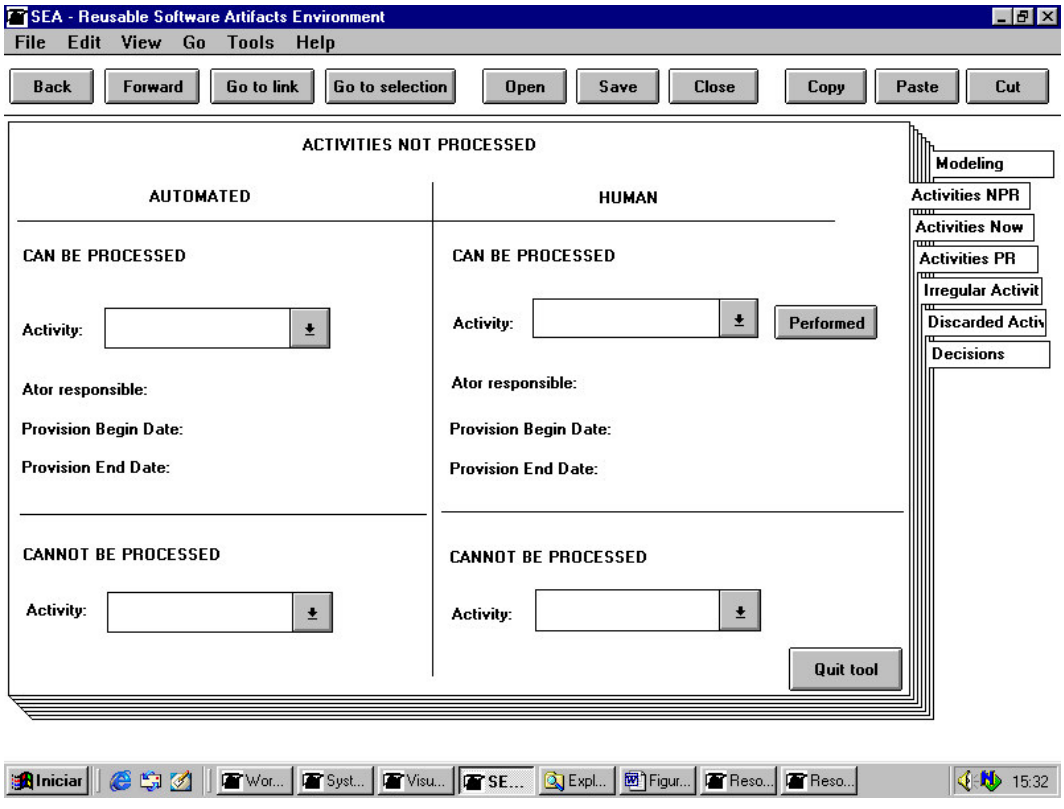



Figura 6.14 – Atividades NPR

Quando as atividades (humanas e automatizadas) estão no estado ‘hão processada”, elas aparecem na página dessa opção do Sistema Gerenciador de *Workflow* As atividades humanas podem ser processadas nesse momento, quando o ator responsável pela atividade, a seleciona e clica no botão “*Performed*”. Nesse instante, essa atividade humana torna-se processada e retorna um arquivo texto sobre o que ela realizou. Em relação às atividades automatizadas, elas apenas aparecem na página, com seus atributos.

É possível visualizar as atividades que podem ser processadas e as atividades que não podem ser processadas naquele instante do projeto.

O Sistema Gerenciador ainda apresenta as decisões que foram tomadas e as decisões que ainda esperam algum julgamento. O item “*Decision*s” mostra as decisões processadas, as decisões descartadas e as decisões que esperam por uma escolha, com as possíveis atividades a serem escolhidas.

O ator responsável pela decisão define a atividade a ser disparada. A figura 6.15 mostra as decisões que aguardam processamento, as decisões processadas e as decisões descartadas.

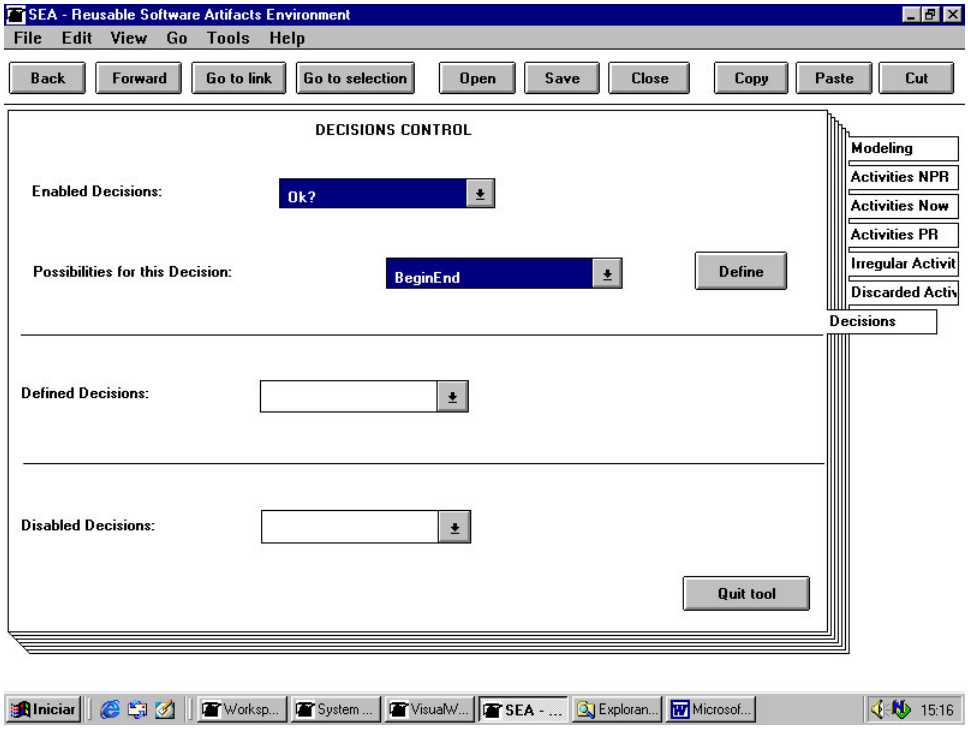



Figura 6.15 – Gerenciamento das Decisões

Assim como as atividades, as decisões também apresentam cores para a identificação de seus estados: verde representa decisão tomada e azul representa decisão a ser tomada. O início e término de processo (*BeginEnd*) também se utiliza de cores para a representação de início de *Workflow* (*begin* fica com a cor verde) e término do projeto (end fica com a cor verde).

### 6.7.1 Janelas do Gerenciador de *Workflow*

As janelas do Gerenciador de *Workflow* apresentam:

- Atividades não processadas: o nome de cada ator responsável pela atividade, data provável de início da atividade e data provável de término da atividade.

- Atividades em andamento, atividades processadas e atividades irregulares: o nome de cada ator responsável pela atividade, a data provável de início da atividade, a data real de início da atividade e a data provável de término da atividade. As atividades processadas e irregulares ainda apresentam a data real de término da atividade.

- Atividades descartadas: o nome do ator responsável pela atividade.

- Decisões: nome das decisões indefinidas (e suas possíveis atividades), nome das decisões descartadas, nome das decisões tomadas.

As Figuras 6.16, 6.17 e 6.18 mostram as telas do Sistema Gerenciador de *Workflow* do ambiente SEA2.

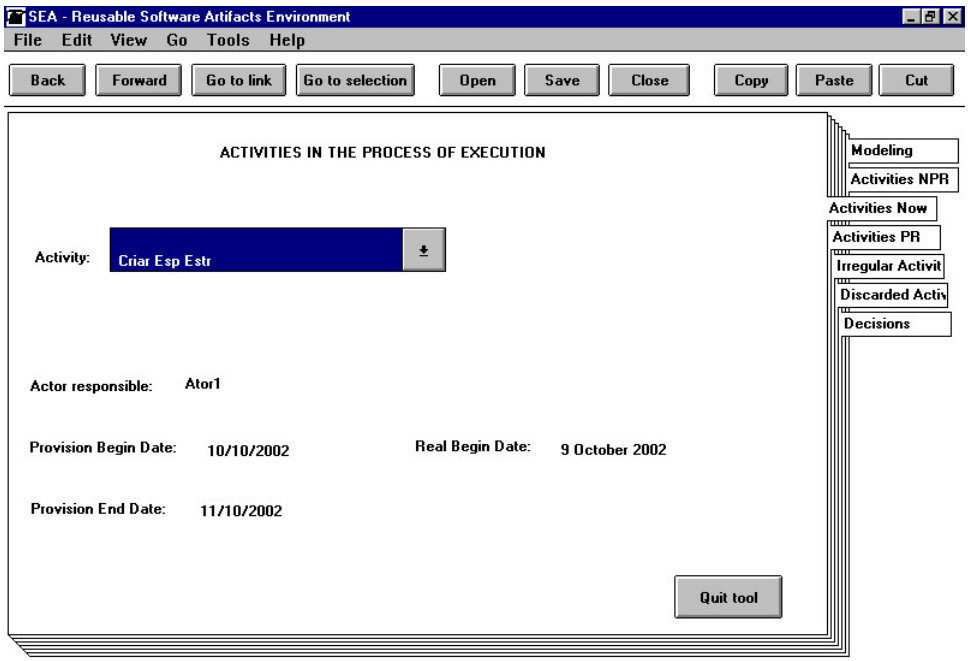



Figura 6.16– Opção *Activities Now*

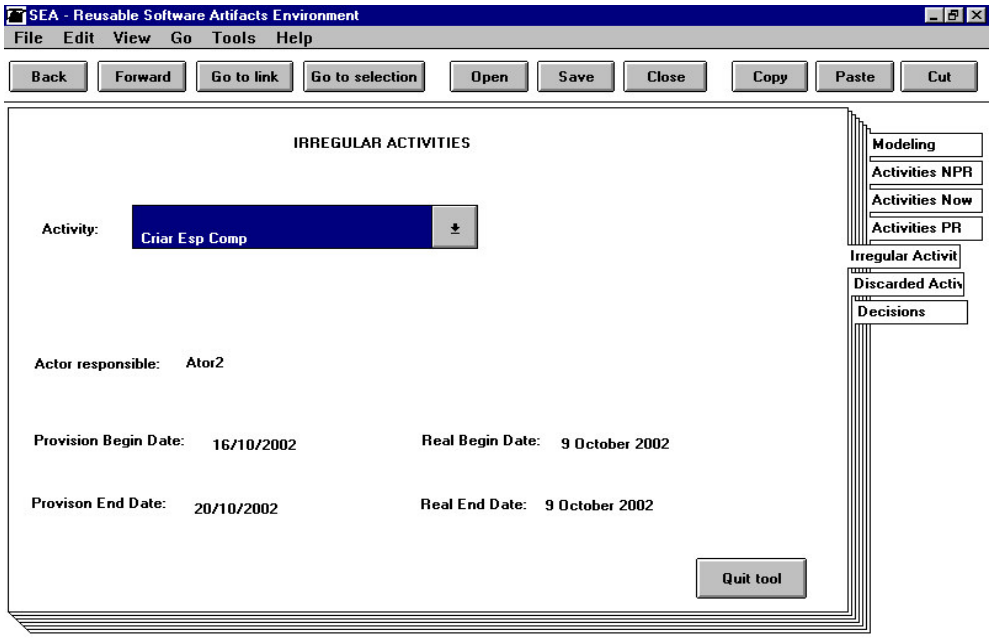



Figura 6.17 – Opção *Activities IR*

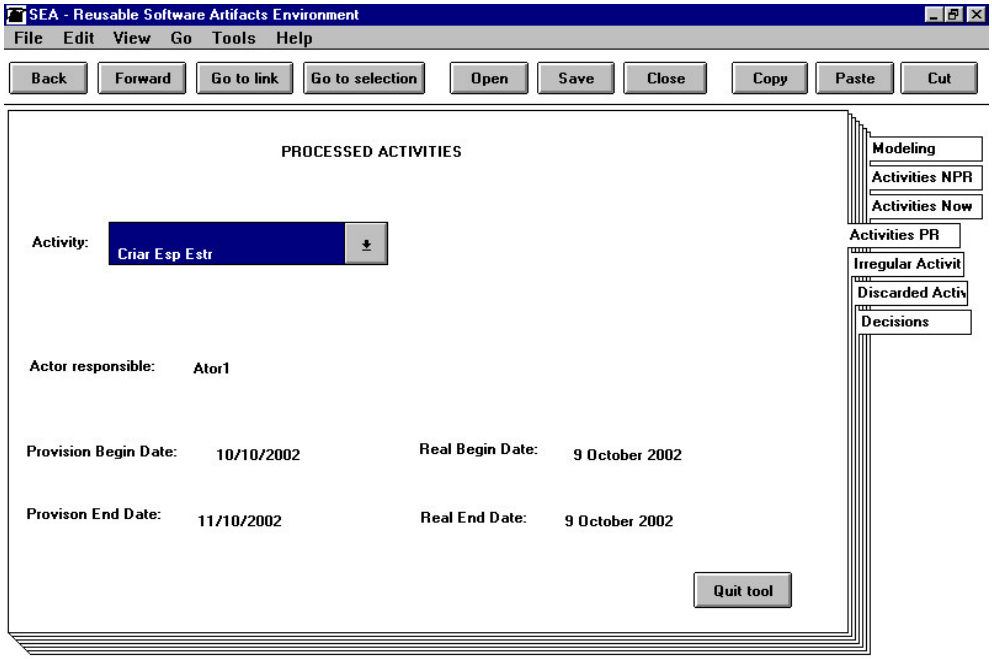



Figura 6.18 – Opção *Activites PR*

### 6.7.2 Consistência do Gerenciador de *Workflow*

Tendo os estados das atividades, o sistema gerenciador pode mostrar aos usuários o desenvolvimento do *Workflow*. É possível visualizar o processo que está ocorrendo no momento, os processos terminados, quais decisões foram tomadas, qual ator ainda não terminou suas atividades, quais atividades estão pendentes e quem é o responsável pelo atraso e, com esses dados, detectar onde estão os problemas do desenvolvimento de um *software*.

O modelo de *Workflow* pode definir e documentar o planejamento do projeto, o planejamento das atividades, os compromissos a serem cumpridos, os atores responsáveis pelas atividades e as estimativas do trabalho a ser realizado (colocadas no *Workflow* através de atributos das atividades). Essas características disponibilizadas pelo ambiente SEA2 estão associadas a KPA Planejamento e Projeto de *Software*.

As ações elaboradas no ambiente SEA2 para suprir essa KPA estão descritas abaixo.

Meta1: Estimativas são documentadas para uso no planejamento e acompanhamento do projeto.

Procedimento realizado no ambiente SEA2 para suprir a Meta1: Ao ser inserida uma atividade no Editor de *Workflow*, ela tem suas datas prováveis de início e término definidas. Assim, sua estimativa de tempo de execução está documentada. Ao definir a primeira atividade, já é possível ter uma previsão de término do projeto e fazer um acompanhamento geral dos passos dos processos. Assim, pode ser estabelecido um cronograma do *Workflow*.

Meta2: Atividades e compromissos (do projeto) são planejados e documentados.

Procedimento realizado no ambiente SEA2 para suprir a Meta2: As atividades e decisões a serem tomadas durante a execução do projeto são criadas e documentadas no Diagrama de *Workflow*.

Meta3: Indivíduos e grupos afetados concordam com seus compromissos relacionados ao projeto.

Procedimento realizado no ambiente SEA2 para suprir a Meta3: As atividades, decisões e compromissos são designados para seus respectivos atores, definidos no Diagrama de *Workflow*. Caso exista alguma discordância de definição de projeto por parte de algum ator, outro ator é designado para a elaboração das tarefas (atividades).

A KPA Acompanhamento e Supervisão do Projeto de *Software* possui algumas metas que precisam de informações a respeito das atividades. Assim, existem atributos nas atividades (como datas pré-estabelecidas que são comparadas com datas reais de início e término de atividades) que auxiliam no gerenciamento e permitem que ações corretivas possam ser tomadas com as atividades que estão desviando-se dos planos previstos. Caso seja necessária alguma alteração, elas podem ser efetuadas no próprio Diagrama. A associação feita entre as metas desta KPA e os procedimentos incluídos no ambiente SEA2 são descritos abaixo.

Meta1: Resultados e performance são acompanhados considerando o planejamento.

Procedimento realizado no ambiente SEA2 para suprir essa meta: O Sistema Gerenciador de *Workflow* permite o acompanhamento do desenvolvimento do projeto, seguindo as atividades e os seus estados. São atribuídas cores aos possíveis estados de cada atividade, portanto, pode-se saber se uma atividade não está processada, se a atividade está em andamento, processada ou irregular. Ao saber a condição da atividade, por exemplo, uma atividade processada, pega-se os atributos referentes às datas prováveis de início e término dessa atividade, e compara-se com as datas reais de início e término da atividade. Assim, tem-se o resultado e, através da comparação entre datas, tem-se a performance do desenvolvimento desta atividade. Caso as datas sejam iguais, o resultado retorna atividade processada no tempo estipulado com desempenho de 100%. Caso contrário, o resultado retorna um erro de cálculo ou inabilidade do ator em executar tal atividade.

Meta2: Ações corretivas são executadas quando os resultados desviam-se dos planos.

Procedimento realizado no ambiente SEA2 para suprir essa meta: Com as datas reais de início e término de uma atividade, pode-se fazer uma comparação com as datas previstas e, caso seja necessário, correções podem ser feitas com as atividades que ainda não foram processadas, para que uma nova estimativa final do produto do desenvolvimento do *software* seja produzida. A alteração (correção) é feita diretamente no Diagrama de *Workflow*.

A KPA Garantia de Qualidade de *Software* permite o acompanhamento e a verificação de conformidade entre o que foi planejado e o que está sendo processado. Existe uma revisão e auditoria das atividades executadas. O ambiente SEA2 proporciona as seguintes metas desta KPA:

Meta1: Representante verifica conformidade das atividades.

Procedimento realizado no ambiente SEA2 para suprir essa meta: Um ator pertencente ao *Workflow* pode visualizar o Gerenciador e o andamento das atividades. Assim, compara com o que foi definido no Diagrama de *Workflow* e o que realmente está sendo processado.

Meta2: Representante reporta resultados obtidos com a auditoria sob as atividades.

Procedimento realizado no ambiente SEA2 para suprir essa meta: Um ator pode relatar as atividades que estão com seu tempo de processamento em conformidade com o que foi planejado e relatar atividades que não estão de acordo com a modelagem prevista.

Meta3: Desvios identificados são documentados e comunicados caso fiquem sem solução.

Procedimento realizado no ambiente SEA2 para suprir essa meta: As atividades consideradas irregulares (e que não foram solucionadas) podem ser salvas junto com a modelagem de *Workflow*. O Diagrama pode guardar atividades com essa condição.

Algumas das KPAs do nível 2 (e de outros níveis) de CMM também podem ser as próprias atividades como, por exemplo, a atividade "Revisão de conformidade entre as atividades anteriores" (ação reparatória pertence a KPA Garantia de Qualidade de *Software)*, onde esta atividade percorre todas as outras anteriores e verifica se nenhum desvio ou erro ocorreu. Neste caso, a ação da atividade é percorrer outras atividades e testar as condições do seu estado atual e comparar com o que foi definido na modelagem inicial.

# 7 EXEMPLO DE PROJETO DESENVOLVIDO NO AMBIENTE SEA2

## 7.1 Descrição

Um exemplo proposto no ambiente SEA2 é o desenvolvimento de um *framework* e de uma aplicação usando esse *framework*. Para o desenvolvimento do *framework* faz-se necessário a criação dos seguintes Diagramas: Diagrama de Casos de Uso, Diagrama de Atividade, Diagrama de Seqüência, Diagrama de Classes, Diagrama de Corpo de Método e Diagrama de Transição de Estados. A modelagem a ser produzida no Ambiente SEA2 para este desenvolvimento, é a seguinte:

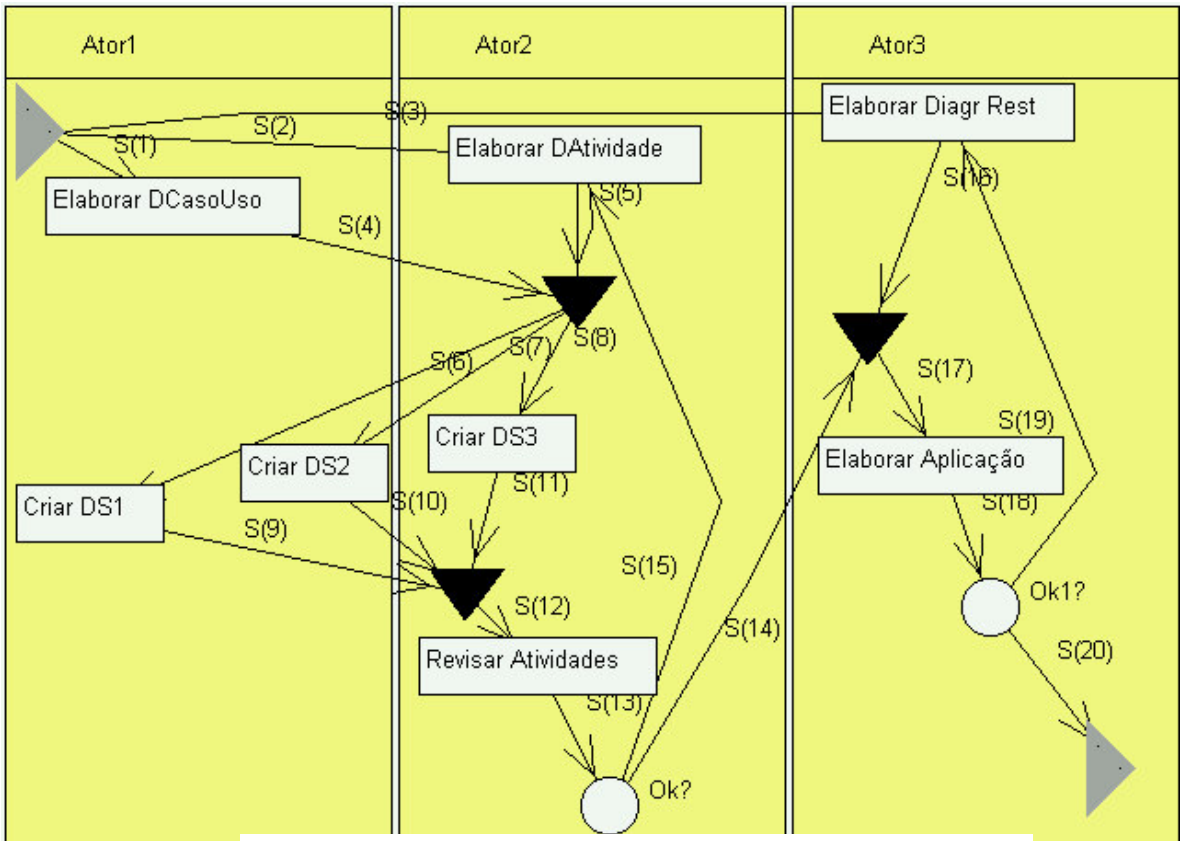



Figura 7.1 – Modelagem do exemplo proposto

O início do *Workflow* ocorre com o processamento de três atividades automatizadas (ligadas através do *link* ao seu documento correspondente): "Elaborar DCasoUso", "Elaborar DAtividade" e "Elaborar Diagr Rest". Após a conclusão das atividades "Elaborar DCasoUso" e "Elaborar DAtividade" é disparado um sincronismo que exige que essas atividades estejam processadas para o disparo das próximas atividades.

A Figura 7.2 mostra as atividades processadas (em verde) e a atividade em andamento (em azul), que ainda não teve seu documento congelado.

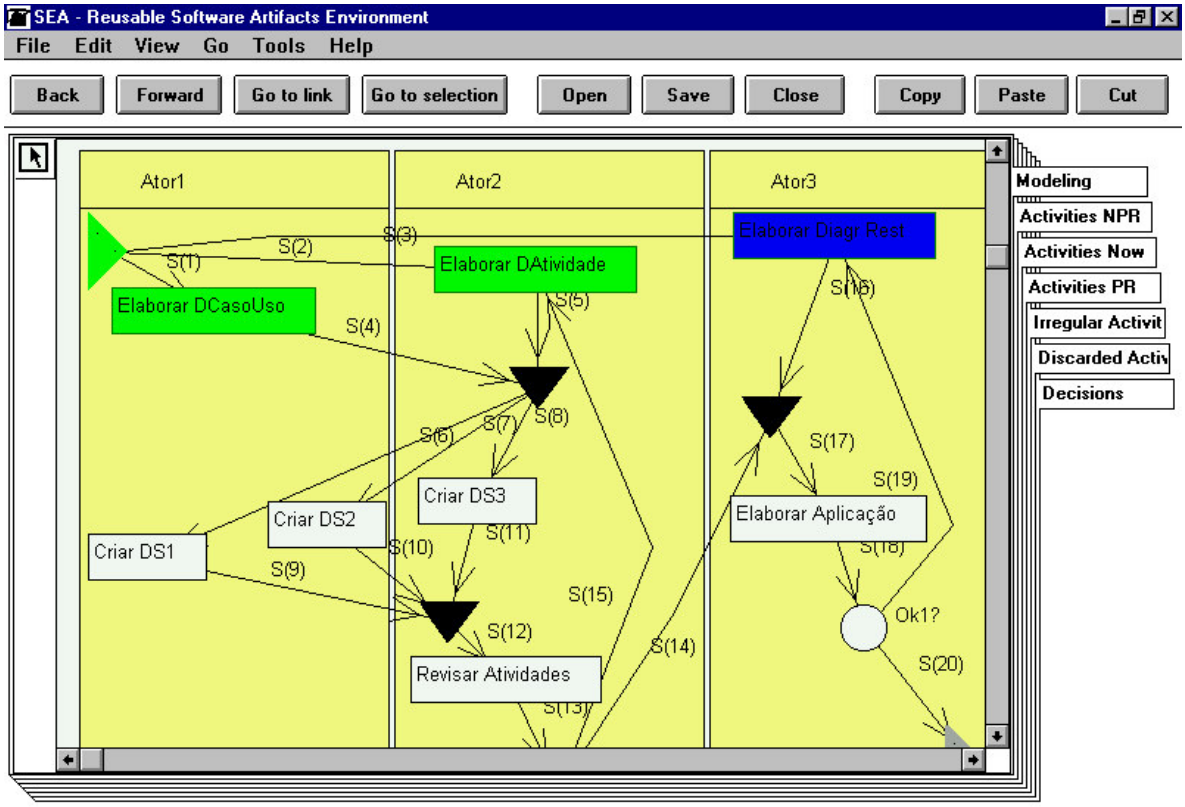



Figura 7.2 – Prosseguimento do *Workflow*

Com o sincronismo validado, as atividades automatizadas "Criar DS1", "Criar DS2" e "Criar DS3" podem ser processadas. Ao serem concluída s, essas atividades também possuem um disparo para um sincronismo. O sincronismo é necessário para

permitir o processamento da próxima atividade, "Revisar Atividades", que é uma atividade humana.

O processamento da atividade humana é realizado na janela do Sistema Gerenciador de *Workflow,* enquanto que o processamento das atividades automatizadas depende do congelamento da especificação associada à atividade.

Após o término da atividade "Revisar Atividades", é disparado um *trigger* a uma decisão, para que seja escolhido o andamento do *Workflow*. A Figura 7.3 mostra esta parte do desenvolvimento.

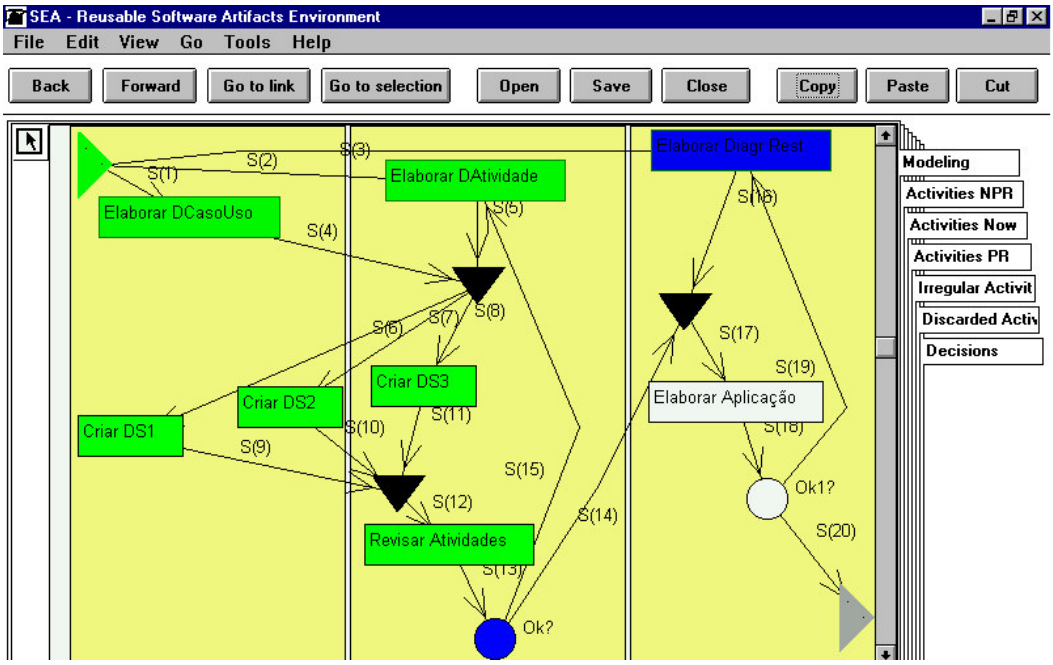



Figura 7.3 – Andamento do *Workflow*

Caso não exista nenhuma inconsistência com as atividades "Elaborar DAtividade", "Criar DS1", "Criar DS2" e "Criar DS3", o andamento dos processos segue conforme planejado. Caso contrário, o andamento do *Workflow* volta para a atividade "Elaborar DAtividade" (Figura 7.4).

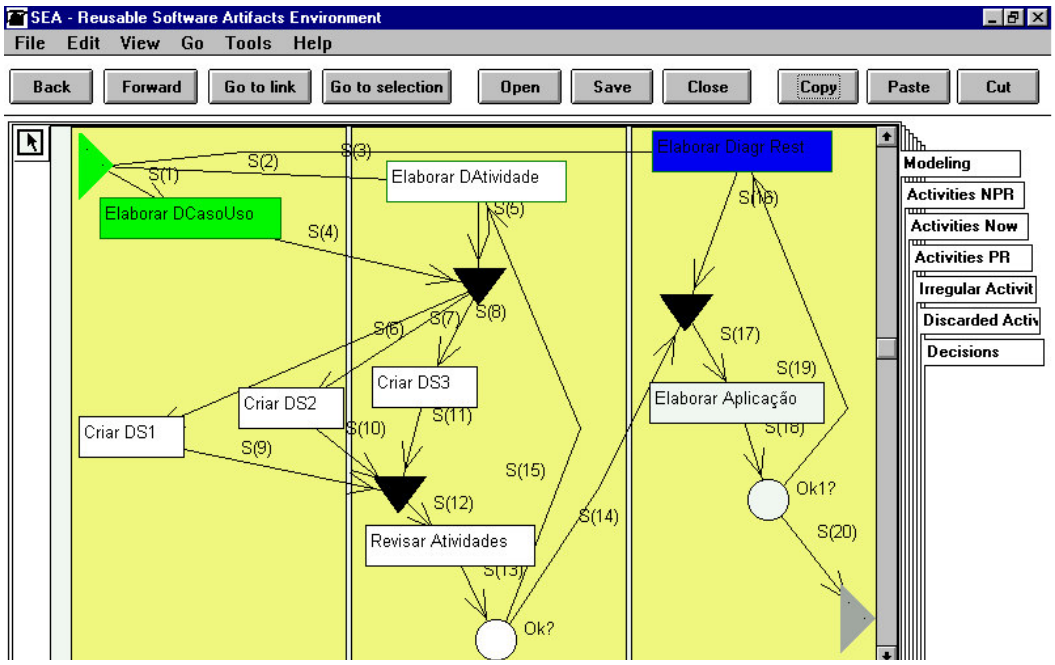



Figura 7.4 – Decisão retorna o processamento

No momento em que o processamento retorna para atividades, sincronismo ou decisão anteriores, todas as atividades posteriores retornam para o estado NPR. Isso ocorre porque a atividade(s) deve ser refeita já que não está processada da maneira correta. O histórico dos *links* anteriormente criados fica guardado nas características da atividade.

Caso não existam problemas com as atividades, é disparado um *trigger* para um sincronismo. Esse sincronismo exige que as atividades processadas pelos atores "Ator1" e "Ator2" e a atividade "Elaborar Diagr Rest" (pertencente ao Ator3) estejam concluídas para a continuidade do processamento. Com a conclusão dessas atividades, é disparado um *trigger* para a atividade "Elaborar Aplicação".

Ao término da atividade "Elaborar Aplicação" é tomado uma decisão que retorna se existe ou não alguma inconformidade com essa atividade. Caso exista, o processamento do *Workflow* retorna para a atividade "Elaborar Especificação". Caso contrário, é terminado o *Workflow* (Figura 7.5).

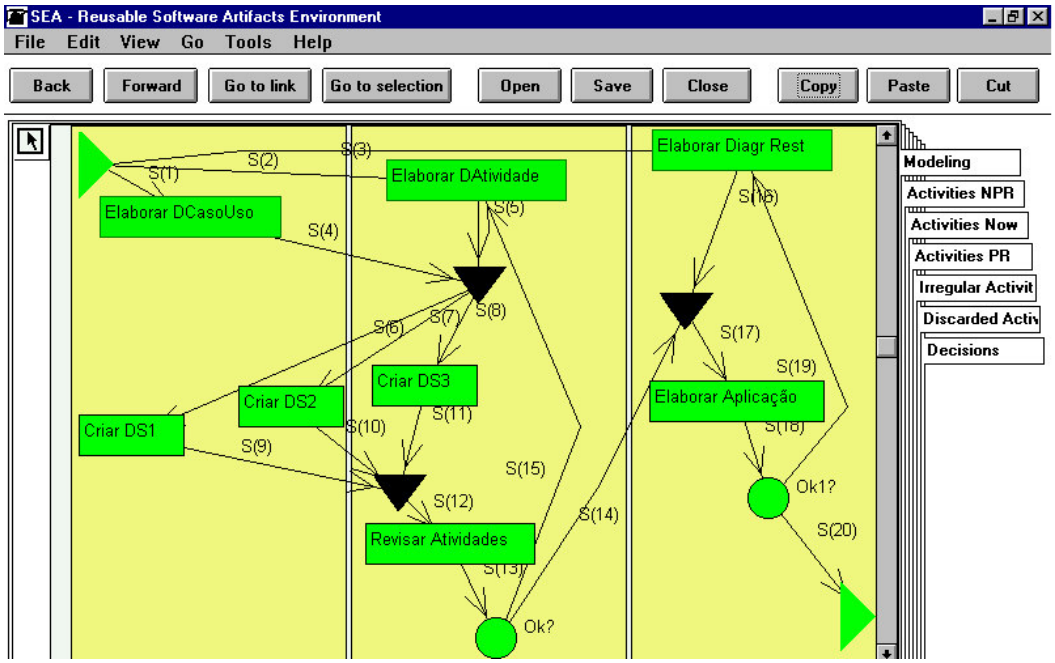



Figura 7.5 – Término do *Workflow*

Para usar a ferramenta de Gerenciamento de *Workflow* (*Workflow Management tool*) e verificar o andamento dos processos, todos os conceitos incluídos no Diagrama devem estar consistentes com suas regras (a ferramenta de Análise de Especificação - SEA Project A*nalyser* verifica isso) e a especificação deve estar congelada (*Frozen*).

As decisões a serem tomadas durante todo o andamento do *Workflow* são efetuadas no Sistema Gerenciador de *Workflow*. A decisão deve ser a escolha entre uma ou outra atividade, dependendo da resposta do Ator responsável pela decisão.

A descrição das atividades e atores é a seguinte:

| **Atividade** | **Participante** |
|---|---|
| Elaborar DCasoUso | Ator1 |
| Criar DS1 | Ator1 |
| Criar DS2 | Ator1 |
| Elaborar DAtividade | Ator2 |
| Criar DS3 | Ator2 |
| Revisar Atividades | Ator2 |
| Elaborar Diagr Rest | Ator3 |
| Elaborar Aplicação | Ator3 |

Tabela 7.1 – Descrição das atividades e atores

As ações de cada atividade são as seguintes:

Atividade: **Elaborar DCasoUso**

Descrição: O Ator1 deve criar O Diagrama de Casos de Uso referente ao *framework*.

Participante: Ator1

S(1): Ação: Criar o Diagrama de Casos de Uso.
Forma de Disparo: Ativar sistema e permitir o início do processo.

Atividade: **Criar DS1**

Descrição: O Ator1 deve verificar e elaborar o primeiro Diagrama de Seqüência referente ao *framework*.

Participante: Ator1

S(6): Ação: Criar o Diagrama de Seqüência.
Forma de Disparo: O sincronismo permitir que esta atividade possa ser processada. Atividades "Elaborar DCasoUso" e "E laborar DAtividade" devem estar processadas (congelada).

Atividade: **Criar DS2**

Descrição: O Ator1 deve verificar e elaborar o segundo Diagrama de Seqüência referente ao *framework*.

Participante: Ator1

S(7): Ação: Criar o Diagrama de Seqüência.
Forma de Disparo: O sincronismo permitir que esta atividade possa ser processada. Atividades "Elaborar DCasoUso" e "Elaborar DAtividade" devem estar processadas (congelada).

Atividade: **Elaborar DAtividade**

Descrição: O Ator2 deve verificar e elaborar o Diagrama de Atividades referente ao *framework* aplicação.

Participante: Ator2

S(2): Ação: Criar o Diagrama de Atividades.
Forma de Disparo: Ativar sistema e permitir que o Ator2 inicie o processo.

Atividade: **Criar DS3**

Descrição: O Ator2 deve criar o terceiro Diagrama de Seqüência referente ao *framework*.

Participante: Ator2

S(8): Ação: Criar o Diagrama de Seqüência.

Forma de Disparo: O sincronismo permitir que esta atividade possa ser processada. Atividades "Elaborar DCasoUso" e "Elaborar DAtivida de" devem estar processadas (congelada).

Atividade: **Revisar Atividades**

Descrição: Esta atividade humana deve conferir as atividades anteriores e verificar se não existe nenhuma inconformidade do que foi processado com o que foi planejado.

Participante: Ator2

S(12): Ação: Avaliação de um gerente sênior.

Forma de Disparo: Atividade "Criar DS1", "Criar DS2" e "Criar DS3" estarem processadas (congelada)

Atividade: **Elaborar Diagr Rest**

Descrição: O Ator3 deve formar todos Diagramas restantes pertencentes à especificação. Entre eles estão os Diagramas de Classes, Diagrama de Corpo Método e Diagrama de Transição de Estados.

Participante: Ator3

S(3): Ação: Criar os diagramas restantes.

Forma de Disparo: Ativar sistema e permitir o início do processo.

Atividade: **Elaborar Aplicação**

Descrição: O Ator3 deve unir todos os processos efetuados anterioremente e elaborar a aplicação.

Participante: Ator3

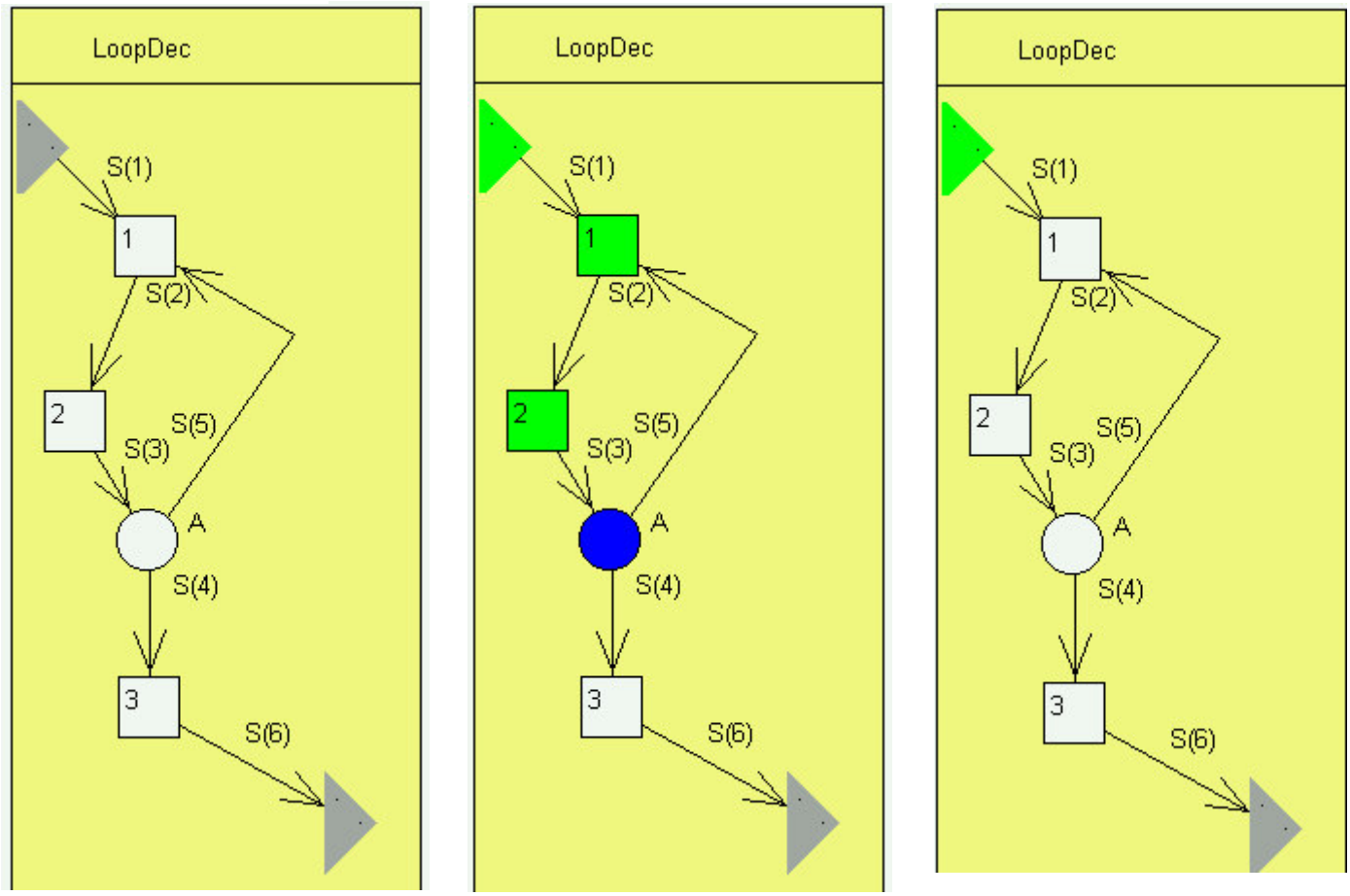
S(16): Ação: Elaborar a aplicação.

Forma de Disparo: Atividades "Elaborar Especificação" e "Revis ar Atividades" estarem processada (congelada).

## 7.2 Situações específicas de processamento

Algumas situações de processamento são esclarecidas nessa Seção para um melhor entendimento do funcionamento.

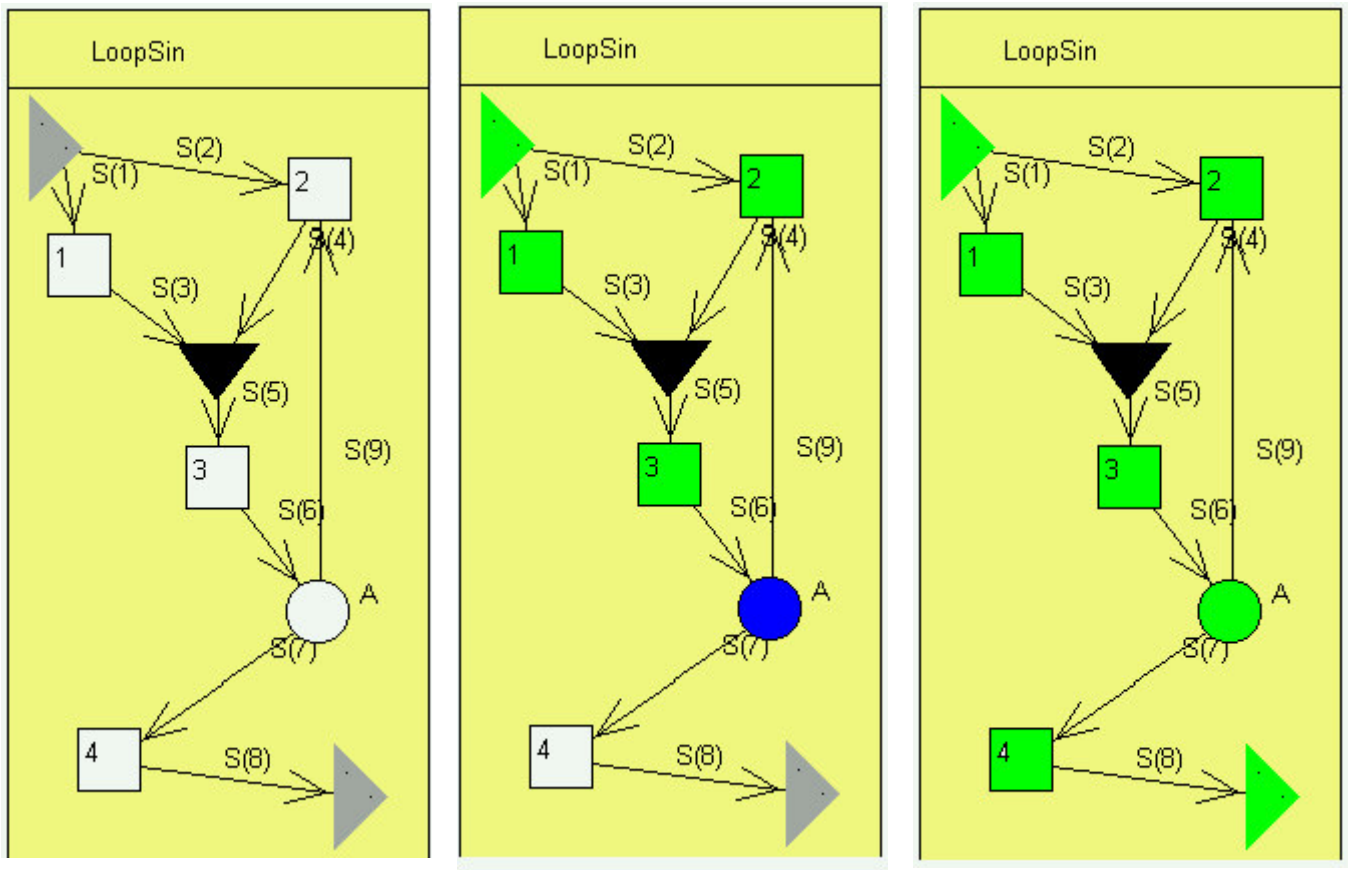
1ª Situação:




Neste caso, após as atividades 1 e 2 serem processadas, uma decisão deve ser tomada. Caso a decisão escolha retornar o processamento para a atividade 1, o andamento do *Workflow* retorna para esta atividade e propaga NPR para as atividades seguintes. A decisão fica novamente desabilitada.

Isso ocorre pelo fato de se o processamento da atividade 1 está inconsistente, as atividades conseguintes também devem (ou não) ser alteradas. Portanto, a atividade 2 deverá ser novamente processada.

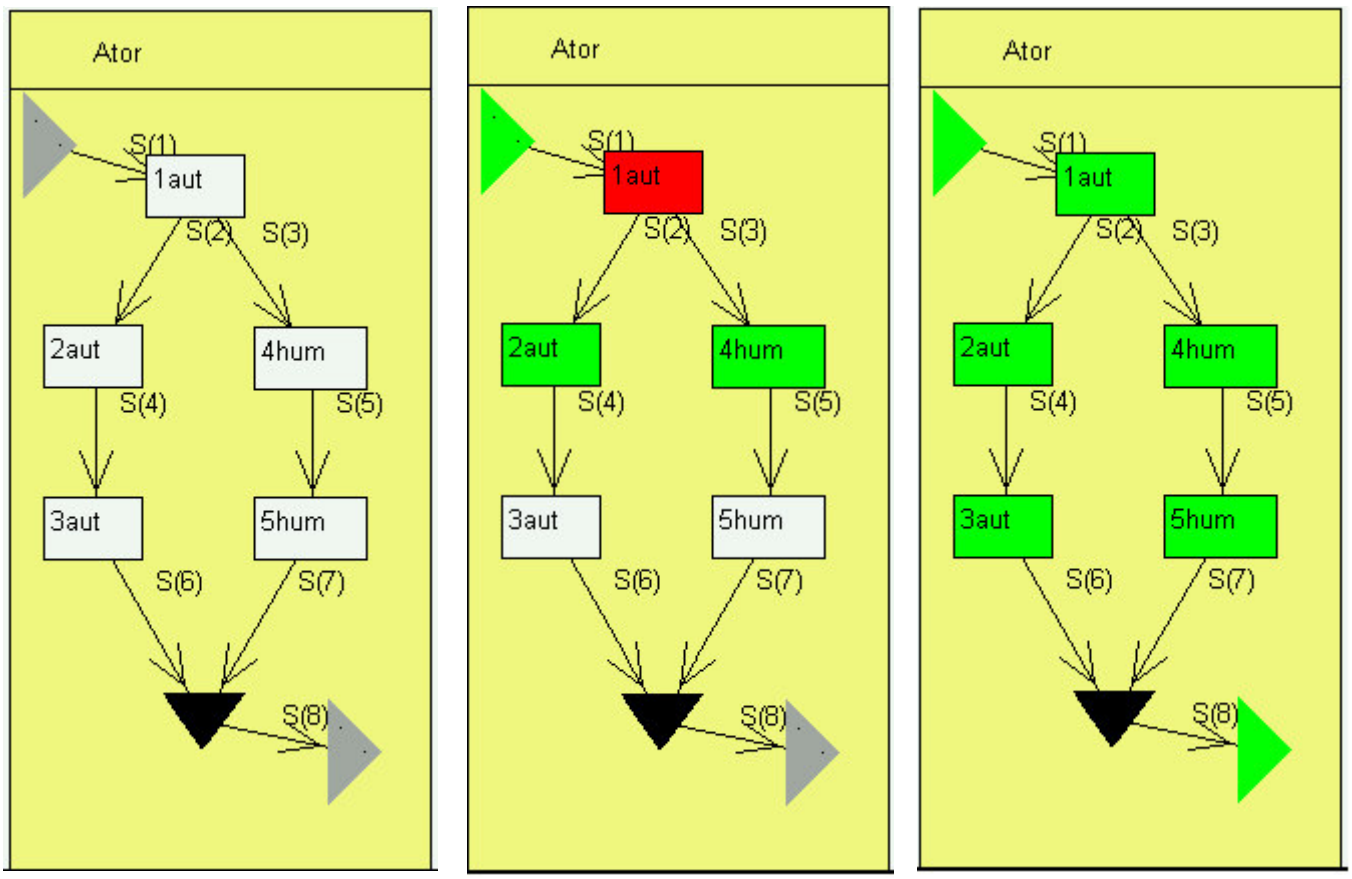
2ª Situação:




Nesta situação, as atividades 1 e 2 precisam ser processadas para a continuidade do *Workflow*.

Após o processamento da atividade 3, é tomada uma decisão. Caso a decisão seja o processamento da atividade 2, o andamento do *Workflow* retorna para esta atividade e torna a atividade 3 como NPR e a decisão como DIS. Caso a decisão resolva processar a atividade 4, as ações tomadas com os outros elementos do modelo permanecem iguais e ocorre o término do *Workflow*.

3ª Situação:




Após a atividade "1aut" ser processada, as atividades "2aut" e "4hum" podem ser processadas. Caso ocorra uma irregularidade com a atividade "1aut", ela se torna uma atividade IR. O processamento das atividades continua, mas o término do *Workflow* só ocorre quando a atividade irregular estiver seu estado novamente PR, ou seja, processado.

## 7.3 Resultados obtidos

Um dos objetivos propostos neste trabalho, a possibilidade de modelar os passos de um desenvolvimento de *software,* foi alcançado com sucesso. O ambiente SEA não proporcionava modelagem de processos nem acompanhamento do desenvolvimento. Com as novas ferramentas existentes, isso é possível.

Com a possibilidade de garantir a integridade dos processos, através do Sistema Gerenciador de *Workflow*, uma das principais características de gerenciamento de processos (conformidade entre o andamento dos processos com o que foi planejado na modelagem) foi suprida por SEA2.

Outra característica de *Workflow* que SEA2 proporcionou foi a possibilidade de verificar o andamento dos processos e problemas referentes aos cumprimentos das datas previstas. Com os atributos data provável de início de processo, data provável de término de processo, data real de início de processo e data real de final de processo, pertencentes a uma atividade, é possível saber (através do Sistema Gerenciador de *Workflow*) se a atividade está atrasada em relação ao seu cronograma previsto.

Alguns pontos positivos pertencentes ao SEA2 podem ser destacados: a possibilidade de gerenciar decisões e a possibilidade de verificação em tempo real das atividades consideradas automatizadas (os estados das atividades são alterados de acordo com estado dos documentos associados às atividades) e das atividades consideradas humanas (os estados dessas atividades são alterados durante o gerenciamento das mesmas).

A verificação dos pontos, acima descritos, é feita pela própria elaboração do exemplo, que sem as novas ferramentas jamais poderia ser implementado.

# 8 CONCLUSÃO

A proposta deste projeto foi a inclusão de elementos de *Workflow* e de um Sistema Gerenciador de *Workflow* no ambiente SEA.

Os capítulos anteriores mostraram as características do antigo ambiente SEA, as vantagens do uso de sistemas de *Workflow* em ambientes de desenvolvimento de *software*, as práticas aplicáveis do nível 2 de CMM e a introdução de gerenciamento no novo ambiente, SEA2.

Os principais resultados obtidos com esse trabalho, suas limitações, futuras extensões e outros caminhos de pesquisa que podem ser seguidos utilizando os resultados do presente trabalho, são descritos a seguir.

## 8.1 Resultados alcançados

Os elementos de *Workflow* foram inseridos através de um Editor, que possibilitou a modelagem e manipulação dos processos durante o desenvolvimento de *software* e, o Sistema Gerenciador de *Workflow* foi incluído em SEA2 como uma ferramenta do ambiente, que seguiu algumas regras do nível 2 de CMM.

Após essas inclusões, algumas características foram observadas no ambiente SEA2: organização dos processos, controle automático do estado dos processos, distribuição das atividades a serem realizadas, acompanhamento do trabalho, controle de consistência de informações e priorização de tarefas.

Com o novo ambiente SEA2 é possível elaborar projetos, controlar seu desenvolvimento e tomar atitudes corretivas, se for necessário.

## 8.2 Limitações

Algumas limitações são encontradas em SEA2:

- Por ser monousuário, SEA2 não permite comunicação entre atores;
- Não existe o uso de permissões (entre atores) para processamento de atividades;
- O ambiente não está preparado para funcionar em rede;
- Não foi inserido o tratamento a exceções (onde existe a possibilidade de especificar quais ações devam ser executadas caso uma tarefa falhe ou o *Workflow* não consiga ser completado). (NICOLAU, 1998)
- Tratamento da semântica de datas e esforços não foi realizado.
- Não existe a possibilidade de conter várias instâncias de um *Workflow* em um projeto.
- O elemento do tipo sincronismo apenas emprega um "*and*" do tipo lógico, ou seja, somente permite a continuação do processamento se todas suas atividades anteriores forem consideradas processadas.
- Impossibilidade de associar um conjunto de documentos a uma única tarefa, o que atrapalha o desenvolvimento de uma especificação OO.

## 8.3 Trabalhos Futuros

Sugere-se que sejam elaborados projetos nas áreas em que este ambiente possui limitações como, por exemplo, torná-lo multiusuário, permitir seu uso em rede, implementar o uso de e-mail entre atores e adaptá-lo à *web*.

Pode-se também, alterar a atual modelagem de *Workflow* para outra, por exemplo, o modelo temporal orientado a objetos, TF-ORM, que permite modelar restrições temporais, trocas dinâmicas e tratamento de exceções. (NICOLAU, 1998, EDELWEISS, 2002), ou o modelo de Redes de Petri (BARROS, 1997) ou o modelo de Casati/Ceri (AMARAL, 1997).

No momento em que outras modelagens poderiam ser usadas no ambiente SEA2, haveria a possibilidade de uma avaliação da modelagem de *Workflow* mais adequada para o desenvolvimento de *softwares*.

Pode-se também incluir outras regras de CMM ao ambiente, permitindo estender os requisitos das KPAs e enriquecer o ambiente quanto à qualidade do *software*.

## 8.4 Considerações Finais

Os ambientes de desenvolvimento de *software* têm buscado melhorar a qualidade e produtividade.

A qualidade nos processos de *software* demanda definição do projeto, descrição de atividades necessárias, métodos e ferramentas a serem utilizadas para cada atividade e avaliação do *software*. Os procedimentos de gerência são importantes, pois monitoram os processos dentro de um projeto. Para isso, uma solução adequada pode ser o uso de *Workflow* juntamente com CMM.

O presente trabalho foi motivado pelo fato de o ambiente SEA não possuir gerenciamento dos processos no desenvolvimento de *software*. Nesse sentido, SEA2 conseguiu proporcionar gerenciamento no desenvolvimento de aplicações e conseguiu cumprir algumas KPAs de CMM, possibilitando assim, o aumento da qualidade dos *software*s desenvolvidos sob ele, segundo preconizado por este modelo de processo.

# REFERÊNCIAS BIBLIOGRÁFICAS


AKKERSDIJK, Victor; BLAAUW, Martin; FAASE, Eric. ***Trigger Modeling for Workflow Analysis***. 1998. Disponível por http://panoramix.univ-paris1.fr/CRINFO/dmrg/MEE98/misop003/index4.html#[JOOS94] Capturado em 03 Novembro 2002

AMARAL, Vinícius L. **Técnicas de Modelagem de *Workflow***. Porto Alegre: CPGCC da UFRGS, 1997. (TI - 622).

AMARAL, Vinícius L.; GRALA, Anderson S.; LIMA, José V. ***Workflow* e Gerência de Documentos.** In: XVII Congresso da Sociedade Brasileira de Computação - XVI Jornada de Atualização em Informática, Brasília, DF: 1997.

ARAUJO, MarcoA. **Automatização do Processo de Desenvolvimento de *Software* nos Ambientes instanciados pela Estação TABA.** Rio de Janeiro: Engenharia de Sistemas e Computação COPPE/UFRJ, Junho 1998. Dissertação de Mestrado.

BARNES, Anthony.; GRAY, Jonathan. ***COTS, Workflow, and Software Process Management: an exploration of software engineering tool development.*** In: IEEE 2000.

BARRETO, José Jr. QUALIDADE DE *SOFTWARE*. 1997. Disponível por http://www.inf.ufsm.br/~oliveira/elc311/qualidadeSW.html#12207 Capturado em 25 Julho 2002.

BARROS, Rodolfo Miranda. *Alocação de Atividades em um Sistema de Gerência de Workflow*. Porto Alegre: Instituto de Informática da Universidade Federal do Rio Grande do Sul, 1997. 116p. Dissertação de Mestrado.

BARTHELMESS, P. **Sistemas de *Workflow*: Análise da Área e Proposta de Modelo**. São Paulo: Instituto de Computação – UNICAMP, 1996. Tese.

BELLUR, Umesh. ***The role of components & standards in software reuse. In: Workshop on Compositional Software Architectures.*** Monterey, California. ***Proceedings…*** Disponível por http://www.objs.com/ workshops/ws9801/>. Capturado em 17 Agosto 1998.

BIZFLOW, Advanced *Software* Solutions for Business Process Management and Automated *Workflow*. Disponível por http:// www.handysoft.com Capturado em 05 Outubro 2002.

BRANT, J. ***HotDraw***. Urbana: University of Illinois at Urbana-Champaign, 1995. Master thesis.

BRITTO, Eduardo; GUANDELINE, Eidy; LOPES, Filipe; MOLZ, Kurt; THOM, Lucinéia. **Levantamento das Ferramentas de *Workflow*.** Porto Alegre: PGCC da UFRGS, 2001.

CARDOSO, Janette; VALETTE, Robert. **Redes de Petri**. Florianópolis: Ed. Da UFSC, 1997. 212p.

CASTILHO, Fernando M. **Distribuição de Tarefas em Sistemas de *Workflow* usando Lógica Nebulosa.** Florianópolis: Programa de Pós-Graduação em Ciência da Computação, Agosto 2002. Dissertação de Mestrado.

CHEN, Yin-Shinn; WANG, Feng-Jian. ***An Editing System for Working Processes.*** In: IEEE 25th *Annual International Computer Software and Applications Conference* (COMPSAC 2001), 2001. ***Proceedings…*** IEEE, 2001.

CORDENONZI, Walkíria H. **Mapeamento de Padrões Internacionais de Qualidade de Produto e de *Software* para um Modelo Conceitual de Gerência do Processo de Desenvolvimento de *Software*.** Porto Alegre: Instituto de Informática da UFRGS, 2000. Dissertação de Mestrado.

COSA, Solutions. Turning *Workflow* into Cash Flow. 2002. Disponível por http://www.cosa.nl Capturado em 05 Outubro 2002.

DEUTSCH, P. ***Frameworks and reuse in the Smalltalk 80 System***. In: BIGGERSTAFF, T. ***Software reusability***. New York: ACM Press, 1989. V.1, p. 57-71.

EDELWEISS, Nina. O Modelo TF-ORM, 1998. Disponível por http://www.inf.ufrgs.br/~nina/indice.htm Capturado em 05 Outubro 2002.

FILENET. The substance behind e Business. 2002. Disponível por http://www.filenet.com Capturado em 05 Outubro 2002.

FORO. Schlumberger Sema, 2002. Disponível por http:// www.sema.es Capturado em 05 Outubro 2002.

HEIMAN, Peter; KRAPP, Carl-Arndt; WESTFECHTEL, Bernhard. ***An Environment for Managing Software Development Processes.*** In: IEEE *8th Conference Software Engineering Environments* (SEE 1997), 1997. ***Proceedings…*** IEEE 1997. p. 101- 108.

KLABUNDE, Charles C. **Tratando *Frameworks* como Componentes**. Porto Alegre: Instituto de Informática da UFRGS, 2000. Dissertação de Mestrado.

KOZAK, Dalton Vinícius. INTRODUÇÃO À QUALIDADE. Disponível por http://www.lami.pucpr.br/~dalton/Apres/intqual/sld040.htm Capturado em 25 Julho 2002.

KROTH, Marcelo L. **Estudo Sobre Sistemas de *Workflow***. Porto Alegre: Instituto de Informática da Universidade Federal do Rio Grande do Sul, 1997. TI.

LARSSON, Johan L. ***Reuse, genericity and frameworks: A graduate thesis in computer science***. ***Proceedings…*** Disponível em CD: FAYAD, Mohamed E; et al. ***Implementing Aplications Frameworks***. New York: Wiley Computer Publishing, 1999.

LARSSON, Johan L. An Overview of EFLIB. ***Proceedings…*** Disponível em CD: FAYAD, Mohamed E; et al. ***Implementing Aplications Frameworks***. New York: Wiley Computer Publishing, 1999.

LIMA, Carla A.G.; REIS, Rodrigo Q.; NUNES, Daltro J. **Gerenciamento do Processo de Desenvolvimento Cooperativo de *Software* no Ambiente PROSOFT.** In: XII Simpósio Brasileiro de Engenharia de *Software*, 1998, p.221-236.

LOTUS, Development Corporation. ***Domino Workflow Key Features and Benefits***. 2000. Disponível por http://www.lotus.com Capturado em 08 Outubro 2002.

MACHADO, Luis F. C. **Modelo para Definição de Processos de *Software* na Estação TABA.** Rio de Janeiro: Engenharia de Sistemas e Computação COPPE/UFRJ, 2000. Dissertação de Mestrado.

NICOLAU, Mariano**. Um Estudo sobre Conceituação de *Workflow*.** Porto Alegre: Instituto de Informática. Porto Alegre: CPGCC da UFRGS, 1996.44p. Monografia (Mestrado em Ciência da Computação).

NICOLAU, Mariano. Modelagem de *Workflow* utilizando um modelo de Dados Temporal Orientado a Objetos com Papéis. Porto Alegre: Instituto de Informática. Porto Alegre: CPGCC da UFRGS, 1998. Dissertação de Mestrado.

ORACLE, Corporation. ***Oracle Workflow – Feature Overview***. 2000. Disponível por http://www.oracle.com Capturado em 08 Outubro 2002.

PASETTI, Alessandro; PREE, Wolfgang. ***Framework Methodology Project – Overview*.** Disponível por http://www.aut.ee.ethz.ch/~pasetti/FrameworkMethodology/ProjectOverview.html Capturado em 03 Novembro 2002

PAULK, Mark., WEBER, Charles, CURTIS, Bill, CHRISSIS, Mary. ***The Capability Maturity Model: Guidelines for Improving the Software Process***. Carnegie Mellon University – *Software* Engineering Institute. 1994.

SETTI, Mariângela et. al. ***Projeto Businebpress*: Um *Workflow* para *Web*, Modular e Integrável**. Curitiba: Universitária Champagnat. In: *International Symposium on Knowlegment Management / Document Management* (ISKM-DM2000), 2000, p.297-316.

SILVA, Ricardo Pereira e. **Suporte ao Desenvolvimento e uso de *Frameworks* e Componentes**. Porto Alegre: Instituto de Informática da UFRGS, 2000a. Tese de Doutorado.

SILVA, Ricardo Pereira e. **Disciplina de Desenvolvimento Orientado a Objetos com *Frameworks*, Componentes e *Patterns*** – INE 6606. Florianópolis: Programa de Pós-Graduação em Ciência da Computação, UFSC, setembro 2000b.

SILVA, Ricardo Pereira e. **Curso de CMM**. Florianópolis: Programa de Pós-Graduação em Ciência da Computação, UFSC, março 2001.

SILVA, Ricardo P. e, PRICE, R.T.. **Suporte ao desenvolvimento e uso de componentes flexíveis**. In: Proceedings of XIII Simpósio Brasileiro de Engenharia de Software. Florianópolis: oct. 1999. p13-28.

TALIGENT. ***Leveraging object-oriented frameworks***. Taligent Inc. white paper, 1995.

TALIGENT, Inc. ***Building Object-Oriented Frameworks*****.** 1997. Disponível por http://lhcb-comp.web.cern.ch/lhcb-comp/Components/postscript/buildingoo.pdf Capturado em 03 Novembro 2002.

THOM, Lucinéia H.; SCHEIDT, Neiva. **Estudo Sobre Modelagem e Aplicação de Sistemas de *Workflow***. Santa Cruz do Sul: Departamento de Informática. Universidade de Santa Cruz do Sul, 1999. Trabalho de Conclusão.

THOM, Lucinéia H. **Associando estrutura organizacional e Modelagem de *Workflow*.** Porto Alegre: PGCC da UFRGS, 2001. TI II.

TIBCO. 2002.. Disponível por http://www.tibco.com Capturado em 05 Outubro 2002.

VERAART, V.E.; WRIGHT, S.L. ***Dukas: A Software Process Task Management Environment.*** In: IEEE *Australian Software Engineering Conference* (ASWEC 1997), 1997, Austrália. ***Proceedings…*** IEEE, 1997. p. 101- 108.

*WORKFLOW* MANAGEMENT COALITION**.** ***Terminology & Glossary***. Bruxelas, Feb. 1999. 65p. Disponível por WWW por http://www.wfmc.org Capturado em 08 Outubro 2002.